LiMo Foundation™

BETA: Subject to revision prior to final release of LiMo Foundation Platform Release 1

Messaging API

Version 1.0
March 6, 2008

PLEASE REVIEW THE TERMS OF THE FOUNDATION API LICENSE VERSION 1.0 BEFORE READING THIS DOCUMENT.

You may view the Foundation API License v1.0 at:
http://www.limofoundation.org/images/stories/pdf/limo_foundation_api_license_v1-0.pdf

This is an agreement between you ("You") and LiMo Foundation related to the use of the APIs contained within this document. By accessing, downloading, reading or otherwise using the APIs contained within this document, You accept the terms of the Foundation API License v1.0. If You do not agree to the terms of this agreement, do not download, read and/or otherwise use any of the APIs.

If You are agreeing to be bound by the Foundation API License v1.0 on behalf of your employer, You represent and warrant that you have full legal authority to bind your employer to this agreement. If You do not have the requisite authority, You may not download, read and/or otherwise use any of the APIs on behalf of your employer.

© Copyright LiMo Foundation 2007, 2008. All rights reserved.

NOTICES

This document and the technical information it contains is subject to change by LiMo Foundation and its members at any time without notice as authorized by LiMo Foundation bylaws and other documents governing LiMo Foundation, including amendments thereto. All statements regarding LiMo's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Any reference in this document to non-LiMo Foundation Web sites do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of this document or any product described in this document and use of those Web sites is at your own risk.

Information concerning non-LiMo products was obtained from the suppliers of those products or a public announcement or source. LiMo cannot confirm the accuracy of that information or the performance, compatibility or any other claims related to non-LiMo products.

All names or other identifying information used in examples in this document are fictitious and any similarity to the names or other identifying information of an actual person or organization is entirely coincidental.

THIS DOCUMENT AND ALL INFORMATION IN THIS DOCUMENT ARE PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED WARRANTY OR CONDITION, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY OR CONDITION OF ACCURACY OR COMPLETENESS, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR AGAINST INFRINGEMENT. ALL EXPRESS AND IMPLIED STATUTORY WARRANTIES ARE DISCLAIMED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE OR NON-INFRINGEMENT.

BETA: Subject to revision prior to final release of LiMo Foundation Platform Release 1

© Copyright LiMo Foundation 2007, 2008. All rights reserved.

Contents

1 Messaging API **1**

1.1 References . . . 1
1.2 Overview . . . 1
1.3 Industry Standard . . . 1
1.4 External API Documents . . . 1
1.5 Plug-in Extension Point Interface . . . 2
1.6 Other Interfaces . . . 2
1.7 Other Notes . . . 2

2 Data Structure Index **3**

2.1 Data Structures . . . 3

3 File Index **5**

3.1 File List . . . 5

4 Page Index **7**

4.1 Related Pages . . . 7

5 Data Structure Documentation **9**

5.1 _MSG_ACCOUNT_ID_LIST_S Struct Reference . . . 9
5.2 _MSG_ACCOUNT_S Struct Reference . . . 10
5.3 _MSG_CONTACT_LIST_S Struct Reference . . . 12
5.4 _MSG_CONTACT_S Struct Reference . . . 13
5.5 _MSG_FILTER_ID_LIST_S Struct Reference . . . 15
5.6 _MSG_FILTER_LIST_S Struct Reference . . . 16
5.7 _MSG_FILTER_S Struct Reference . . . 17
5.8 _MSG_FOLDER_ID_LIST_S Struct Reference . . . 19
5.9 _MSG_FOLDER_S Struct Reference . . . 20
5.10 _MSG_MESSAGE_ID_LIST_S Struct Reference . . . 22
5.11 _MSG_MESSAGE_REQUEST_S Struct Reference . . . 23

5.12 _MSG_MESSAGE_S Struct Reference 25
5.13 _MSG_MESSAGE_STATUS_S Struct Reference 29
5.14 _MSG_MESSAGE_TYPE_LIST_S Struct Reference 31
5.15 _MSG_MESSAGE_TYPE_S Struct Reference 32
5.16 _MSG_QUERY_ID_LIST_S Struct Reference 33
5.17 _MSG_QUERY_LIST_S Struct Reference 34
5.18 _MSG_QUERY_S Struct Reference 35
5.19 _MSG_SORT_RULE_S Struct Reference 36
5.20 _MSG_SORT_TYPE_LIST_S Struct Reference 37
5.21 _MSG_STORAGE_ID_LIST_S Struct Reference 38
5.22 _MSG_STORAGE_REGISTRY_S Struct Reference 39
5.23 _MSG_STRING_S Struct Reference 40

6 File Documentation **41**

6.1 ComponentOverview.h File Reference 41
6.2 Msg_Mapi_Account.h File Reference 42
6.3 Msg_Mapi_Contact.h File Reference 43
6.4 Msg_Mapi_FilterRule.h File Reference 44
6.5 Msg_Mapi_Folder.h File Reference 47
6.6 Msg_Mapi_Message.h File Reference 48
6.7 Msg_Mapi_Message_Request.h File Reference 49
6.8 Msg_Mapi_Message_Status.h File Reference 50
6.9 Msg_Mapi_QueryRule.h File Reference 51
6.10 Msg_Mapi_Session.h File Reference 54
6.11 Msg_Mapi_Storage_Account.h File Reference 66
6.12 Msg_Mapi_Storage_Folder.h File Reference 77
6.13 Msg_Mapi_Storage_Manager.h File Reference 92
6.14 Msg_Mapi_Storage_Message.h File Reference 103
6.15 Msg_Mapi_Storage_Registry.h File Reference 118
6.16 Msg_Mapi_Types.h File Reference 119

7 Page Documentation **133**

7.1 Overview of the LiMo Messaging API 133
7.2 Sample Code for Using the LiMo Messaging API 140
7.3 Extending a Message Structure 141
7.4 Message Session: Sending a Message 142
7.5 Message Session: Receiving Messages 144

7.6 Storage General: Initializing Storage . . . 146
7.7 Message Storage: Creating and Storing a Message . . . 147
7.8 Message Storage: Retrieving Message List and Entity . . . 149
7.9 Folder Storage: Creating and Storing a Folder . . . 151
7.10 Folder Storage: Querying Folders . . . 152
7.11 Account Storage: Creating and Storing an Account . . . 153
7.12 Account Storage: Updating an Account . . . 155
7.13 Account Storage: Getting a Default Account . . . 156
7.14 Sequence Charts for the LiMo Messaging API . . . 157
7.15 Messaging Framework Startup . . . 159
7.16 Session: Sending a Message . . . 160
7.17 Session: Receiving Messages . . . 161
7.18 Message Storage: Creating and Opening Storage Connection . . . 162
7.19 Message Storage: Closing and Destroying Storage Connection . . . 163
7.20 Message Storage: Creating and Storing A Message . . . 164
7.21 Message Storage: Querying Messages . . . 165
7.22 Folder Storage: Creating and Storing a Folder . . . 166
7.23 Account Storage: Creating and Storing an Account . . . 167
7.24 Filter Plug-in: Set Filter List to Filter Messages . . . 168
7.25 Sending Messages (Normal Flow) . . . 169
7.26 Sending Messages (Alternative Flow) . . . 170
7.27 Receiving Messages . . . 171
7.28 Explore Message List (Flow for Listing Messages) . . . 172
7.29 Explore Message List (Flow for Browsing a Message) . . . 173
7.30 Explore Message List (Flow for Removal of a Message) . . . 174
7.31 Explore Message List (Flow for Moving Messages Between Folders) . . . 175
7.32 Manage Account and Profile (Flow for Creation of an Account) . . . 176
7.33 Manage Account and Profile (Flow for Update of an Account) . . . 177
7.34 Manage Account and Profile (Setting Default Account) . . . 178
7.35 Manage Account and Profile (Flow for Removing an Account) . . . 179
7.36 Manage Account and Profile (Alternative Flow for Removing an Account) . . . 180

Chapter 1

Messaging API

1.1 References

[1] *Software Architecture Document*, Version 1.1 Revision 1.4, LiMo Foundation, 11 September 2007

[2] *Messaging Framework API*, Version 1.0, LiMo Foundation, 1 February 2008

1.2 Overview

The LiMo Messaging API (MAPI) provides services to send, receive, parse and assemble messages over arbitrary message delivery mechanisms such as MMS, SMS, email, and i-mode message. The Messaging services consist of common functions and sets of service-specific functions. MAPI provides a collection of data structures as well as functions to implement the services. The data structures and functions are grouped into modules, as listed below:

- Message Rending and Reception
- Storage Management
- Message Storage
- Folder Storage
- Account Storage

1.3 Industry Standard

None

1.4 External API Documents

None

1.5 Plug-in Extension Point Interface

None

1.6 Other Interfaces

None

1.7 Other Notes

The following supplemental material is also available in this document.

- Overview of the LiMo Messaging MAPI
- Sample Code for Using the LiMo Messaging API
- Sequence Charts for the LiMo Messaging API

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

_MSG_ACCOUNT_ID_LIST_S (Represents an account id list) . . . 9
_MSG_ACCOUNT_S (Represents an account) . . . 10
_MSG_CONTACT_LIST_S (Represents a contact list) . . . 12
_MSG_CONTACT_S (Represents a contact) . . . 13
_MSG_FILTER_ID_LIST_S (Represents a filter id list) . . . 15
_MSG_FILTER_LIST_S (Represents a filter list) . . . 16
_MSG_FILTER_S (Represents a filter) . . . 17
_MSG_FOLDER_ID_LIST_S (Represents a folder id list) . . . 19
_MSG_FOLDER_S (Represents a folder) . . . 20
_MSG_MESSAGE_ID_LIST_S (Represents a message id list) . . . 22
_MSG_MESSAGE_REQUEST_S (Represents a request that will be sent to messaging framework) . . . 23
_MSG_MESSAGE_S (Represents a message) . . . 25
_MSG_MESSAGE_STATUS_S (Represents a message status) . . . 29
_MSG_MESSAGE_TYPE_LIST_S (Represents a message types list) . . . 31
_MSG_MESSAGE_TYPE_S (Represents a message type) . . . 32
_MSG_QUERY_ID_LIST_S (Represents a query Id list) . . . 33
_MSG_QUERY_LIST_S (Represents a query list) . . . 34
_MSG_QUERY_S (Represents a query rule) . . . 35
_MSG_SORT_RULE_S (Represents a sort rule) . . . 36
_MSG_SORT_TYPE_LIST_S (Represents a sort type list) . . . 37
_MSG_STORAGE_ID_LIST_S (Represents a storage id list) . . . 38
_MSG_STORAGE_REGISTRY_S (Represents a storage registry entry) . . . 39
_MSG_STRING_S (Represents a string) . . . 40

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

ComponentOverview.h (Component Programming Interface Overview Header Template for Doxygen) . . . 41
Msg_Mapi_Account.h (Defines the account structure) . . . 42
Msg_Mapi_Contact.h (Defines the contact structure) . . . 43
Msg_Mapi_FilterRule.h (Defines the filter related structures) . . . 44
Msg_Mapi_Folder.h (Defines the folder related structures) . . . 47
Msg_Mapi_Message.h (Defines the message structure) . . . 48
Msg_Mapi_Message_Request.h (Defines the message request structure) . . . 49
Msg_Mapi_Message_Status.h (Defines the message status structure) . . . 50
Msg_Mapi_QueryRule.h (Defines the query related structure) . . . 51
Msg_Mapi_Session.h (Defines session handling and message sending and receiving related functions) . . . 54
Msg_Mapi_Storage_Account.h (Defines account related functions) . . . 66
Msg_Mapi_Storage_Folder.h (Defines folder related functions in MAPI) . . . 77
Msg_Mapi_Storage_Manager.h (Defines storage management functions in MAPI) . . . 92
Msg_Mapi_Storage_Message.h (Defines message storage related functions in MAPI) . . . 103
Msg_Mapi_Storage_Registry.h (Defines the storage registry structure) . . . 118
Msg_Mapi_Types.h (Defines general types) . . . 119

Chapter 4

Page Index

4.1 Related Pages

Here is a list of all related documentation pages:

Overview of the LiMo Messaging API 133
Sample Code for Using the LiMo Messaging API 140
Sequence Charts for the LiMo Messaging API 157

Chapter 5

Data Structure Documentation

5.1 _MSG_ACCOUNT_ID_LIST_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.1.1 Detailed Description

Represents an account id list.

Data Fields

- int iCount
- MSG_ACCOUNT_ID_T * accountIds

5.1.2 Field Documentation

5.1.2.1 int _MSG_ACCOUNT_ID_LIST_S::iCount

The count of message ids

5.1.2.2 MSG_ACCOUNT_ID_T* _MSG_ACCOUNT_ID_LIST_S::accountIds

The pointer to the account id list

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

5.2 _MSG_ACCOUNT_S Struct Reference

`#include <Msg_Mapi_Account.h>`

5.2.1 Detailed Description

Represents an account.

All the fields except for `pExData` contain common information for an account. The field `pExData` points to the user customized structure. This customized structure should be used by application and message storage plugin.

For the more detailed information of account, please refer to Account. This structure is extendable. Please refer to Extendability .

Data Fields

- MSG_ACCOUNT_ID_T accountId
- MSG_STRING_S accountName
- MSG_MESSAGE_TYPE_S msgType
- MSG_FOLDER_ID_T rootFolderId
- MSG_STORAGE_ID_T storageId
- MSG_STORAGE_ID_T defaultMsgStorageId
- size_t extDataSize
- void * pExtData

5.2.2 Field Documentation

5.2.2.1 MSG_ACCOUNT_ID_T _MSG_ACCOUNT_S::accountId

Indicates the unique account index.

5.2.2.2 MSG_STRING_S _MSG_ACCOUNT_S::accountName

Indicates the display name for this account.

5.2.2.3 MSG_MESSAGE_TYPE_S _MSG_ACCOUNT_S::msgType

Indicates the message type of an account, such as SMS, MMS, EMAIL and i-mode message.

5.2.2.4 MSG_FOLDER_ID_T _MSG_ACCOUNT_S::rootFolderId

Indicates the root folder which is assigned to this account.

5.2.2.5 MSG_STORAGE_ID_T _MSG_ACCOUNT_S::storageId

Indicates the storage id in which this account (metadata and extension data) is stored.

5.2.2.6 MSG_STORAGE_ID_T _MSG_ACCOUNT_S::defaultMsgStorageId

Indicates default storage id for the message of this account

5.2.2.7 size_t _MSG_ACCOUNT_S::extDataSize

Indicates the size of `pExtData`

5.2.2.8 void* _MSG_ACCOUNT_S::pExtData

Used to add account extension data. For example, the mail box setting of Email account.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Account.h

5.3 _MSG_CONTACT_LIST_S Struct Reference

`#include <Msg_Mapi_Contact.h>`

5.3.1 Detailed Description

Represents a contact list.

Data Fields

- int iCount
- MSG_CONTACT_S * pContacts

5.3.2 Field Documentation

5.3.2.1 int _MSG_CONTACT_LIST_S::iCount

Indicates the number of the contacts in the list.

5.3.2.2 MSG_CONTACT_S* _MSG_CONTACT_LIST_S::pContacts

The pointer to the contact list.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Contact.h

5.4 _MSG_CONTACT_S Struct Reference

`#include <Msg_Mapi_Contact.h>`

5.4.1 Detailed Description

Represents a contact.

This structure includes the contact basic information. It can present various contact type such as telephone number, email address and so on.

This structure is extendable. Please refer to Extendability .

Data Fields

- MSG_CONTACT_TYPE_T contactType
- unsigned char * contactVal
- MSG_STRING_S contactName
- size_t extDataSize
- void * pExtData

5.4.2 Field Documentation

5.4.2.1 MSG_CONTACT_TYPE_T _MSG_CONTACT_S::contactType

Indicates contact type.

Two types are predefined:

MSG_CONTACT_TYPE_EMAIL is for email address like `abc@example.com`

MSG_CONTACT_TYPE_PLMN is for phone number like +10123456789

5.4.2.2 unsigned char* _MSG_CONTACT_S::contactVal

The value of the contact. It should be encoded in ASCII.

For example, the real phone number or email address.

5.4.2.3 MSG_STRING_S _MSG_CONTACT_S::contactName

Indicates the display name of this contact.

5.4.2.4 size_t _MSG_CONTACT_S::extDataSize

Indicates the size of the extension data which `pExtData` point to. The unit is byte.

5.4.2.5 void* _MSG_CONTACT_S::pExtData

Extension data for contact.

Some data that is not covered in the structure can be put here, such as picture data and so on.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Contact.h

5.5 _MSG_FILTER_ID_LIST_S Struct Reference

`#include <Msg_Mapi_FilterRule.h>`

5.5.1 Detailed Description

Represents a filter id list.

It is composed of various filter ids.

Data Fields

- int iCount
- MSG_FILTER_ID_T * filterIds

5.5.2 Field Documentation

5.5.2.1 int _MSG_FILTER_ID_LIST_S::iCount

Indicates the number of the filter id in the list.

5.5.2.2 MSG_FILTER_ID_T* _MSG_FILTER_ID_LIST_S::filterIds

Indicates the pointer to the filter id list.

The documentation for this struct was generated from the following file:

- Msg_Mapi_FilterRule.h

5.6 _MSG_FILTER_LIST_S Struct Reference

`#include <Msg_Mapi_FilterRule.h>`

5.6.1 Detailed Description

Represents a filter list.

It is composed of various MSG_FILTER_S. The first filter in the list has the highest priority. Framework will check the filter list from the first filter to the last one. Once the message satisfies the filter, framework stops checking other filters in the list and returns the `autoStore` and `intercept` properties of the current filter.

Data Fields

- int iCount
- MSG_FILTER_S * filters

5.6.2 Field Documentation

5.6.2.1 int _MSG_FILTER_LIST_S::iCount

Indicates the number of the filters in the list.

5.6.2.2 MSG_FILTER_S* _MSG_FILTER_LIST_S::filters

The pointer to the filter list.

The documentation for this struct was generated from the following file:

- Msg_Mapi_FilterRule.h

5.7 _MSG_FILTER_S Struct Reference

`#include <Msg_Mapi_FilterRule.h>`

5.7.1 Detailed Description

Represents a filter.

It represents the basic filter unit.

Data Fields

- MSG_FILTER_ID_T filterId
- MSG_STRING_S filterName
- bool autoStore
- bool intercept
- int valueLength
- unsigned char * filterValue

5.7.2 Field Documentation

5.7.2.1 MSG_FILTER_ID_T _MSG_FILTER_S::filterId

Indicates the filter type.

The filter Id indicates the filter type. There are some predefined filter ids. Please refer to _MSG_FILTER_-ID_E

5.7.2.2 MSG_STRING_S _MSG_FILTER_S::filterName

Indicates the display name for the filter.

5.7.2.3 bool _MSG_FILTER_S::autoStore

Indicates whether the message should be automatically stored by framework, when it satisfies this filter.

If it is set `true`, the message will be automatically stored by framework when the message satisfies this filter. By default, this message will be saved to Inbox in the default account.

If it is set `false`, the message will not be stored by framework.

5.7.2.4 bool _MSG_FILTER_S::intercept

Indicates whether the message should be intercepted, when it satisfies this filter.

When the message satisfies this filter, framework will send the message to this filter associated session, then check the `intercept`:

If it is set true, the sessions whose priority are lower than this filter associated session will not receive this message.

If it is set false, framework will send the message to all the other sessions that are interested in this message.

For details about session priority, please refer to MsgSetPriority.

5.7.2.5 int _MSG_FILTER_S::valueLength

Indicates the length of the filterValue.

5.7.2.6 unsigned char* _MSG_FILTER_S::filterValue

The value of the filter.

For example, if the filter Id show that the message should be filtered by message type, the value maybe SMS, MMS.

The documentation for this struct was generated from the following file:

- Msg_Mapi_FilterRule.h

5.8 _MSG_FOLDER_ID_LIST_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.8.1 Detailed Description

Represents a folder id list.

Data Fields

- int iCount
- MSG_FOLDER_ID_T * folderIds

5.8.2 Field Documentation

5.8.2.1 int _MSG_FOLDER_ID_LIST_S::iCount

The count of message ids

5.8.2.2 MSG_FOLDER_ID_T* _MSG_FOLDER_ID_LIST_S::folderIds

The pointer to the folder id list

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

5.9 _MSG_FOLDER_S Struct Reference

`#include <Msg_Mapi_Folder.h>`

5.9.1 Detailed Description

Represents a folder.

For the more detailed information of folder, please refer to Folder . This structure is extendable. Please refer to Extendability .

Data Fields

- MSG_FOLDER_ID_T folderId
- MSG_FOLDER_ID_T parentId
- MSG_STRING_S folderName
- MSG_STORAGE_ID_T storageId
- MSG_SORT_TYPE_T sortType
- MSG_FOLDER_TYPE_T folderType
- size_t commExtDataSize
- void * pCommExtData

5.9.2 Field Documentation

5.9.2.1 MSG_FOLDER_ID_T _MSG_FOLDER_S::folderId

Indicates folder id.

5.9.2.2 MSG_FOLDER_ID_T _MSG_FOLDER_S::parentId

Indicates parent folder id.

5.9.2.3 MSG_STRING_S _MSG_FOLDER_S::folderName

Indicates the folder name.

5.9.2.4 MSG_STORAGE_ID_T _MSG_FOLDER_S::storageId

Indicates the storage id in which this folder (metadata and extension data) is stored.

5.9.2.5 MSG_SORT_TYPE_T _MSG_FOLDER_S::sortType

Indicates the sort type for the children nodes under this folder.

5.9.2.6 MSG_FOLDER_TYPE_T _MSG_FOLDER_S::folderType

Indicates the folder type which can be Inbox/Outbox...

5.9.2.7 size_t _MSG_FOLDER_S::commExtDataSize

Indicates the size of `pCommExtData`

5.9.2.8 void* _MSG_FOLDER_S::pCommExtData

Used to add extension data for framework storage plugin.

For example, the extension data can be picture, max dispalying size and so on.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Folder.h

5.10 _MSG_MESSAGE_ID_LIST_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.10.1 Detailed Description

Represents a message id list.

Data Fields

- int iCount
- MSG_MESSAGE_ID_T * msgIds

5.10.2 Field Documentation

5.10.2.1 int _MSG_MESSAGE_ID_LIST_S::iCount

The count of message ids

5.10.2.2 MSG_MESSAGE_ID_T* _MSG_MESSAGE_ID_LIST_S::msgIds

The pointer to the message id list

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

5.11 _MSG_MESSAGE_REQUEST_S Struct Reference

`#include <Msg_Mapi_Message_Request.h>`

5.11.1 Detailed Description

Represents a request that will be sent to messaging framework.

Applications compose a request with this structure and send it to message framework via a connected session. This request Id is used to manage the request by framework.

This structure is extendable. Please refer to Extendability .

Data Fields

- MSG_REQUEST_TYPE_T reqType
- MSG_MESSAGE_TYPE_S msgType
- MSG_REQUEST_ID_T reqId
- MSG_MESSAGE_S * pMsg
- size_t extDataSize
- void * pExtData

5.11.2 Field Documentation

5.11.2.1 MSG_REQUEST_TYPE_T _MSG_MESSAGE_REQUEST_S::reqType

Indicates the type of request.

Some predefined types: MSG_REQUEST_TYPE_SEND, MSG_REQUEST_TYPE_SENDSAVE

5.11.2.2 MSG_MESSAGE_TYPE_S _MSG_MESSAGE_REQUEST_S::msgType

Indicates the message type.

Indicates the type of the message such as SMS, MMS, Email etc.

5.11.2.3 MSG_REQUEST_ID_T _MSG_MESSAGE_REQUEST_S::reqId

Indicates the reqeust Id, the unique Id of the request.

When applications use MsgSubmitReq to submit a request to framework, this value will be set by framework and returned to applications. Applications should keep this id to trace the request's status or cancel the request later.

5.11.2.4 MSG_MESSAGE_S* _MSG_MESSAGE_REQUEST_S::pMsg

The pointer to message structure.

When applications submit sending message request, this pointer shall point to the message that should be sent.

5.11.2.5 size_t _MSG_MESSAGE_REQUEST_S::extDataSize

Indicates the size of the extension data that `pExtData` point to. The unit is byte.

5.11.2.6 void* _MSG_MESSAGE_REQUEST_S::pExtData

Used to add extension data for transport plugin.

For example, this field can be used to extend new request for specific service.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Message_Request.h

5.12 _MSG_MESSAGE_S Struct Reference

`#include <Msg_Mapi_Message.h>`

5.12.1 Detailed Description

Represents a message.

This structure includes the common information of vaious specific service message, such as SMS, MMS, Email and i-mode message.

This structure is extendable. Please refer to Extendability .

Data Fields

- MSG_MESSAGE_ID_T msgId
- MSG_FOLDER_ID_T parentId
- MSG_MESSAGE_TYPE_S msgType
- MSG_ACCOUNT_ID_T accountId
- MSG_STORAGE_ID_T storageId
- MSG_STRING_S subject
- MSG_CONTACT_S from
- MSG_CONTACT_LIST_S to
- MSG_STRING_S displaySubject
- MSG_STRING_S displayFrom
- MSG_STRING_S displayTo
- time_t displayTime
- size_t size
- MSG_STATUS_T status
- MSG_STATUS_T deliveryStatus
- bool hasAttachment
- bool hasDrmContent
- MSG_MESSAGE_PRIORITY_T priority
- size_t commExtDataSize
- void * pCommExtData
- size_t extDataSize
- void * pExtData

5.12.2 Field Documentation

5.12.2.1 MSG_MESSAGE_ID_T _MSG_MESSAGE_S::msgId

Indicates the message Id of this message.

5.12.2.2 MSG_FOLDER_ID_T _MSG_MESSAGE_S::parentId

Indicates the parent folder ID.

The parent is a folder.

5.12.2.3 MSG_MESSAGE_TYPE_S _MSG_MESSAGE_S::msgType

Indicates the message type.

Indicates the type of the message such as SMS, MMS, Email, i-mode etc.

5.12.2.4 MSG_ACCOUNT_ID_T _MSG_MESSAGE_S::accountId

Indicates which account the message associated.

5.12.2.5 MSG_STORAGE_ID_T _MSG_MESSAGE_S::storageId

Indicates where this message (metadata and extension data) is stored.

If it is set MSG_STORAGE_DEFAULT, the message will be saved according to `defaultMsgStorageId` in its associated account.Otherwise, the message will be saved according this value.

5.12.2.6 MSG_STRING_S _MSG_MESSAGE_S::subject

Indicates the subject of the message.

5.12.2.7 MSG_CONTACT_S _MSG_MESSAGE_S::from

Indicates where the message comes from.

5.12.2.8 MSG_CONTACT_LIST_S _MSG_MESSAGE_S::to

Indicates where the message will send to.

5.12.2.9 MSG_STRING_S _MSG_MESSAGE_S::displaySubject

Indicates the display content for `subject`.

5.12.2.10 MSG_STRING_S _MSG_MESSAGE_S::displayFrom

Indicates the display content for `from`.

5.12.2.11 MSG_STRING_S _MSG_MESSAGE_S::displayTo

Indicates the display content for `to`.

5.12.2.12 time_t _MSG_MESSAGE_S::displayTime

Indicates the display content for time.

For example, if the message is an incoming message, this value should be the time when the message arrived. Framework is responsible to set this value in this case.

If the message is an outgoing message, this value should be the time when the message be sent out. Application is responsible to set this value in this case.

5.12.2.13 size_t _MSG_MESSAGE_S::size

Indicates the size of the message. The unit is byte.

5.12.2.14 MSG_STATUS_T _MSG_MESSAGE_S::status

Indicates the message's local status, such as read/unread/replied/forwarded/protected ...

5.12.2.15 MSG_STATUS_T _MSG_MESSAGE_S::deliveryStatus

Indicates the delivery status of the message.

5.12.2.16 bool _MSG_MESSAGE_S::hasAttachment

Indicates whether the message has attachment.

If the message has attachment, it should be set `true`.

If the message does not have attachment, it should be set `false`.

5.12.2.17 bool _MSG_MESSAGE_S::hasDrmContent

Indicates whether the message attaches the DRM contents.

If it is set `true`, the message attaches the DRM contents.

If it is set `false`, the message does not attaches the DRM contents.

5.12.2.18 MSG_MESSAGE_PRIORITY_T _MSG_MESSAGE_S::priority

Indicates the priority of this message.

5.12.2.19 size_t _MSG_MESSAGE_S::commExtDataSize

Indicates the size of the data that `pCommExtData` point to. The unit is byte.

5.12.2.20 void* _MSG_MESSAGE_S::pCommExtData

Used to add extension data for framework storage plugin.

This field is the extension data for framework storage plugin. For example, whether the message is replied can be regarded as an common extension attribute.

5.12.2.21 size_t _MSG_MESSAGE_S::extDataSize

Indicates the size of the data that `pExtData` point to. The unit is byte.

5.12.2.22 void* _MSG_MESSAGE_S::pExtData

Used to add extension data for message storage/transport plugin.

This field is the extension data for storage/transport plugin. It is the place to put such message payload information as body and so on.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Message.h

5.13 _MSG_MESSAGE_STATUS_S Struct Reference

`#include <Msg_Mapi_Message_Status.h>`

5.13.1 Detailed Description

Represents a message status.

This stucture contains the information about the status of the request that application has submitted to framework.

This structure is extendable. Please refer to Extendability .

Data Fields

- MSG_REQUEST_ID_T reqId
- MSG_ERROR_T error
- MSG_STATUS_T status
- bool isFinal
- MSG_MESSAGE_ID_T msgId
- size_t extDataSize
- void * pExtData

5.13.2 Field Documentation

5.13.2.1 MSG_REQUEST_ID_T _MSG_MESSAGE_STATUS_S::reqId

Indicates the corresponding request Id.

5.13.2.2 MSG_ERROR_T _MSG_MESSAGE_STATUS_S::error

Indicates the error code.

5.13.2.3 MSG_STATUS_T _MSG_MESSAGE_STATUS_S::status

Indicates the status of the corresponding request.

5.13.2.4 bool _MSG_MESSAGE_STATUS_S::isFinal

Indicates the request is finished or not. If the request is finished, no more status of this request will be sent to client and the request information will be removed from messaging framework.

5.13.2.5 MSG_MESSAGE_ID_T _MSG_MESSAGE_STATUS_S::msgId

Indicates the saved message id, it's applicable when status is MSG_STATUS_SAVED_NOTSENT or MSG_STATUS_SENT_SAVED.

5.13.2.6 size_t _MSG_MESSAGE_STATUS_S::extDataSize

Indicates the size of the extension data that `pExtData` point to. The unit is byte.

5.13.2.7 void* _MSG_MESSAGE_STATUS_S::pExtData

Used to add extension data for transport plugin.

For example, this field can be used to extend new status for specific service.

The documentation for this struct was generated from the following file:

- Msg_Mapi_Message_Status.h

5.14 _MSG_MESSAGE_TYPE_LIST_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.14.1 Detailed Description

Represents a message types list.

Data Fields

- int iCount
- MSG_MESSAGE_TYPE_S * pMsgTypes

5.14.2 Field Documentation

5.14.2.1 int _MSG_MESSAGE_TYPE_LIST_S::iCount

The count of message types

5.14.2.2 MSG_MESSAGE_TYPE_S* _MSG_MESSAGE_TYPE_LIST_S::pMsgTypes

The pointer to message types list

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

5.15 _MSG_MESSAGE_TYPE_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.15.1 Detailed Description

Represents a message type.

This structure includes main type and sub type.

Data Fields

- MSG_MAIN_TYPE_T mainType
- MSG_SUB_TYPE_T subType

5.15.2 Field Documentation

5.15.2.1 MSG_MAIN_TYPE_T _MSG_MESSAGE_TYPE_S::mainType

Message main type

5.15.2.2 MSG_SUB_TYPE_T _MSG_MESSAGE_TYPE_S::subType

Message sub type

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

5.16 _MSG_QUERY_ID_LIST_S Struct Reference

`#include <Msg_Mapi_QueryRule.h>`

5.16.1 Detailed Description

Represents a query Id list.

Data Fields

- int iCount
- MSG_QUERY_ID_T * queryIds

5.16.2 Field Documentation

5.16.2.1 int _MSG_QUERY_ID_LIST_S::iCount

Indciates the count of the query Id in the list

5.16.2.2 MSG_QUERY_ID_T* _MSG_QUERY_ID_LIST_S::queryIds

Indicates the pointer to the query Id list

The documentation for this struct was generated from the following file:

- Msg_Mapi_QueryRule.h

5.17 _MSG_QUERY_LIST_S Struct Reference

`#include <Msg_Mapi_QueryRule.h>`

5.17.1 Detailed Description

Represents a query list.

Data Fields

- int iCount
- MSG_QUERY_S * queries

5.17.2 Field Documentation

5.17.2.1 int _MSG_QUERY_LIST_S::iCount

Indciates the count of the query rules in the list

5.17.2.2 MSG_QUERY_S* _MSG_QUERY_LIST_S::queries

Indicates the pointer to the query list

The documentation for this struct was generated from the following file:

- Msg_Mapi_QueryRule.h

5.18 _MSG_QUERY_S Struct Reference

`#include <Msg_Mapi_QueryRule.h>`

5.18.1 Detailed Description

Represents a query rule.

When applications need to query all SMS with subject containing some values, a MSG_QUERY_ID_T can be added as "MSG_QUERY_SMS_SUBJECT_CONTAIN".

Refer to the following code:

```
MSG_QUERY_S queryRulel;
queryRule.queryId = MSG_QUERY_BY_SUBJECT;
queryRule.queryValue = "SearchingValue";
queryRule.valueLength = strlen(queryRule.queryValue);
```

Data Fields

- MSG_QUERY_ID_T queryId
- int valueLength
- unsigned char * queryValue

5.18.2 Field Documentation

5.18.2.1 MSG_QUERY_ID_T _MSG_QUERY_S::queryId

Indicates what kind of rules should be taken when framework filters the messages

5.18.2.2 int _MSG_QUERY_S::valueLength

Indicates the length of the queryValue

5.18.2.3 unsigned char* _MSG_QUERY_S::queryValue

Indicates the value of the query

The documentation for this struct was generated from the following file:

- Msg_Mapi_QueryRule.h

5.19 _MSG_SORT_RULE_S Struct Reference

`#include <Msg_Mapi_QueryRule.h>`

5.19.1 Detailed Description

Represents a sort rule.

The sort rule structure includes sort type and sort order (ascending or descending). Applications can use the sort rule when querying messages.

Data Fields

- MSG_SORT_TYPE_T sortType
- bool isAscending

5.19.2 Field Documentation

5.19.2.1 MSG_SORT_TYPE_T _MSG_SORT_RULE_S::sortType

Indicates the sort type

5.19.2.2 bool _MSG_SORT_RULE_S::isAscending

Indicates the sort order: ascending or descending

The documentation for this struct was generated from the following file:

- Msg_Mapi_QueryRule.h

5.20 _MSG_SORT_TYPE_LIST_S Struct Reference

`#include <Msg_Mapi_QueryRule.h>`

5.20.1 Detailed Description

Represents a sort type list.

Data Fields

- int iCount
- MSG_SORT_TYPE_T * sortTypes

5.20.2 Field Documentation

5.20.2.1 int _MSG_SORT_TYPE_LIST_S::iCount

Indciates the count of the sort type in the list

5.20.2.2 MSG_SORT_TYPE_T* _MSG_SORT_TYPE_LIST_S::sortTypes

Indicates the pointer to the sort type list

The documentation for this struct was generated from the following file:

- Msg_Mapi_QueryRule.h

5.21 _MSG_STORAGE_ID_LIST_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.21.1 Detailed Description

Represents a storage id list.

Data Fields

- int iCount
- MSG_STORAGE_ID_T * pStorageIds

5.21.2 Field Documentation

5.21.2.1 int _MSG_STORAGE_ID_LIST_S::iCount

The count of storage ids

5.21.2.2 MSG_STORAGE_ID_T* _MSG_STORAGE_ID_LIST_S::pStorageIds

The pointer to storage id list

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

5.22 _MSG_STORAGE_REGISTRY_S Struct Reference

`#include <Msg_Mapi_Storage_Registry.h>`

5.22.1 Detailed Description

Represents a storage registry entry.

For details of the storage, please refer to Storage .

Data Fields

- MSG_MEDIA_TYPE_T media_type
- MSG_STORAGE_ID_T storeId
- bool canDeregister
- unsigned char * name
- unsigned char * path

5.22.2 Field Documentation

5.22.2.1 MSG_MEDIA_TYPE_T _MSG_STORAGE_REGISTRY_S::media_type

Indicates the storage media

5.22.2.2 MSG_STORAGE_ID_T _MSG_STORAGE_REGISTRY_S::storeId

Indicates the unique storage id

5.22.2.3 bool _MSG_STORAGE_REGISTRY_S::canDeregister

Indicates whether the storage can be deregistered

5.22.2.4 unsigned char* _MSG_STORAGE_REGISTRY_S::name

Indicates the name of the storage. It's encoded in ASCII

5.22.2.5 unsigned char* _MSG_STORAGE_REGISTRY_S::path

Indicates the path where the storage is located. It is encoded in ASCII

The documentation for this struct was generated from the following file:

- Msg_Mapi_Storage_Registry.h

5.23 _MSG_STRING_S Struct Reference

`#include <Msg_Mapi_Types.h>`

5.23.1 Detailed Description

Represents a string.

Data Fields

- MSG_ENCODE_TYPE_T encodeType
- int length
- unsigned char * pString

5.23.2 Field Documentation

5.23.2.1 MSG_ENCODE_TYPE_T _MSG_STRING_S::encodeType

The encoding type of the string

5.23.2.2 int _MSG_STRING_S::length

The length of string value

5.23.2.3 unsigned char* _MSG_STRING_S::pString

The string value

The documentation for this struct was generated from the following file:

- Msg_Mapi_Types.h

Chapter 6

File Documentation

6.1 ComponentOverview.h File Reference

6.1.1 Detailed Description

Component Programming Interface Overview Header Template for Doxygen.

This file contains configuration information for documentation generation; it does not contain any API-specific information.

Defines

- #define COMPONENT_OVERVIEW_H

6.1.2 Define Documentation

6.1.2.1 #define COMPONENT_OVERVIEW_H

6.2 Msg_Mapi_Account.h File Reference

6.2.1 Detailed Description

Defines the account structure.

```
#include <stdlib.h>
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Folder.h"
```

Data Structures

- struct _MSG_ACCOUNT_S

  Represents an account.

Typedefs

- typedef struct _MSG_ACCOUNT_S MSG_ACCOUNT_S

6.2.2 Typedef Documentation

6.2.2.1 typedef struct _MSG_ACCOUNT_S MSG_ACCOUNT_S

6.3 Msg_Mapi_Contact.h File Reference

6.3.1 Detailed Description

Defines the contact structure.

```
#include <stdlib.h>
#include "Msg_Mapi_Types.h"
```

Data Structures

- struct _MSG_CONTACT_S

  Represents a contact.

- struct _MSG_CONTACT_LIST_S

  Represents a contact list.

Typedefs

- typedef struct _MSG_CONTACT_S MSG_CONTACT_S
- typedef struct _MSG_CONTACT_LIST_S MSG_CONTACT_LIST_S

6.3.2 Typedef Documentation

6.3.2.1 typedef struct _MSG_CONTACT_LIST_S MSG_CONTACT_LIST_S

6.3.2.2 typedef struct _MSG_CONTACT_S MSG_CONTACT_S

6.4 Msg_Mapi_FilterRule.h File Reference

6.4.1 Detailed Description

Defines the filter related structures.

MSG_FILTER_S and MSG_FILTER_LIST_S define the criteria for session's interested incoming messages.

MSG_FILTER_S represents the basic filter unit, with a pair of filter type and filter value. A filter is satisfied when the filter value meets the requirement of specific filter type.

MSG_FILTER_LIST_S is composed of a series of various MSG_FILTER_S. A filter list is satisfied when one of filters in the list satisfied.

The first filter in the list has the highest priority, the last the lowest. Framework will check the filter list from the first to the last. Once an incoming message satisfies one filter in the list, framwork would stop checking and return.

The framework sends the satisfying messages to filter-list associated applications. If the intercept is set true, the lower priority session would not receive the messages.

See MsgPlgFilterCheckRules in *Messaging Framework API* [2].

```
#include "Msg_Mapi_Types.h"
```

Data Structures

- struct _MSG_FILTER_ID_LIST_S

  Represents a filter id list.

- struct _MSG_FILTER_S

  Represents a filter.

- struct _MSG_FILTER_LIST_S

  Represents a filter list.

Typedefs

- typedef int MSG_FILTER_ID_T

  Represents the filter Id.

- typedef struct _MSG_FILTER_ID_LIST_S MSG_FILTER_ID_LIST_S
- typedef struct _MSG_FILTER_S MSG_FILTER_S
- typedef struct _MSG_FILTER_LIST_S MSG_FILTER_LIST_S

Enumerations

- enum _MSG_FILTER_ID_E {

  MSG_FILTER_BY_UNKNOWN = 0, MSG_FILTER_BY_MSG_TYPE, MSG_FILTER_BY_FROM, MSG_FILTER_BY_HAS_ATTACHMENT,

MSG_FILTER_BY_HAS_DRM_CONTENT, MSG_FILTER_BY_SIZE_GT, MSG_FILTER_BY_SIZE_LT, MSG_FILTER_BY_PRIORITY_GT,

MSG_FILTER_BY_PRIORITY_LT, MSG_FILTER_BY_USER_DEF_BEGIN = 0x00001000 }

Represents the values of filter id.

6.4.2 Typedef Documentation

6.4.2.1 typedef struct _MSG_FILTER_ID_LIST_S MSG_FILTER_ID_LIST_S

6.4.2.2 typedef int MSG_FILTER_ID_T

Represents the filter Id.

A filter Id represents a unique filter type. The values for this type SHOULD be in _MSG_FILTER_ID_E or other extended filter Ids.

6.4.2.3 typedef struct _MSG_FILTER_LIST_S MSG_FILTER_LIST_S

6.4.2.4 typedef struct _MSG_FILTER_S MSG_FILTER_S

6.4.3 Enumeration Type Documentation

6.4.3.1 enum _MSG_FILTER_ID_E

Represents the values of filter id.

This enum is used as the value of MSG_FILTER_ID_T.

For example:

```
MSG_FILTER_S filter;
filter.filterId = MSG_FILTER_BY_MSG_TYPE;
filter.filterValue = (unsigned char*)malloc(sizeof(MSG_MESSAGE_TYPE_S));

MSG_MESSAGE_TYPE_S* msgType = (MSG_MESSAGE_TYPE_S*)filter.filterValue;
msgType->mainType = MSG_MESSAGE_TYPE_SMS;
msgType->subType = MSG_MESSAGE_TYPE_SMS_TEXT;

//set the filter
...
...
//the session will only receive SMS.
```

Enumerator:

MSG_FILTER_BY_UNKNOWN Unknown sort type.

MSG_FILTER_BY_MSG_TYPE The messages whose message type are `filterValue` will be filtered out. filterValue should be MSG_MESSAGE_TYPE_S type.

MSG_FILTER_BY_FROM The messages that come from `filterValue` will be filtered out. filterValue should be char * type, which is encoded in ASCII.

MSG_FILTER_BY_HAS_ATTACHMENT Filter by whether the message has attachment.

If `filterValue` is set 0, the messages that do not has attachment will be filtered out.

If `filterValue` is set 1, the messages that have attachment will be filtered out.

MSG_FILTER_BY_HAS_DRM_CONTENT Filter by whether the message has DRM content.

If `filterValue` is set 0, the messages that do not has DRM content will be filtered out.

If `filterValue` is set 1, the messages that have DRM content will be filtered out.

MSG_FILTER_BY_SIZE_GT The messages whose size are greater than or equal the `filterValue` will be filtered out. filterValue should be size_t type

MSG_FILTER_BY_SIZE_LT The messages whose size are less than or equal the `filterValue` will be filtered out. filterValue should be size_t type

MSG_FILTER_BY_PRIORITY_GT The messages whose priority are greater than or equal the `filterValue` will be filtered out. filterValue should be MSG_MESSAGE_PRIORITY_T type.

MSG_FILTER_BY_PRIORITY_LT The messages whose priority are less than or equal the `filterValue` will be filtered out. filterValue should be MSG_MESSAGE_PRIORITY_T type.

MSG_FILTER_BY_USER_DEF_BEGIN User defined filter type begin value.

6.5 Msg_Mapi_Folder.h File Reference

6.5.1 Detailed Description

Defines the folder related structures.

For the detail information of folder , please refer to Folder .

```
#include <stdlib.h>
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_QueryRule.h"
```

Data Structures

- struct _MSG_FOLDER_S

  Represents a folder.

Typedefs

- typedef struct _MSG_FOLDER_S MSG_FOLDER_S

6.5.2 Typedef Documentation

6.5.2.1 typedef struct _MSG_FOLDER_S MSG_FOLDER_S

6.6 Msg_Mapi_Message.h File Reference

6.6.1 Detailed Description

Defines the message structure.

```
#include <stdlib.h>
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Contact.h"
#include "time.h"
```

Data Structures

- struct _MSG_MESSAGE_S

  Represents a message.

Typedefs

- typedef struct _MSG_MESSAGE_S MSG_MESSAGE_S

6.6.2 Typedef Documentation

6.6.2.1 typedef struct _MSG_MESSAGE_S MSG_MESSAGE_S

6.7 Msg_Mapi_Message_Request.h File Reference

6.7.1 Detailed Description

Defines the message request structure.

```
#include <stdlib.h>
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Message.h"
```

Data Structures

- struct _MSG_MESSAGE_REQUEST_S

  Represents a request that will be sent to messaging framework.

Typedefs

- typedef struct _MSG_MESSAGE_REQUEST_S MSG_MESSAGE_REQUEST_S

6.7.2 Typedef Documentation

6.7.2.1 typedef struct _MSG_MESSAGE_REQUEST_S MSG_MESSAGE_REQUEST_S

6.8 Msg_Mapi_Message_Status.h File Reference

6.8.1 Detailed Description

Defines the message status structure.

```
#include "stdlib.h"
#include "Msg_Mapi_Types.h"
```

Data Structures

- struct _MSG_MESSAGE_STATUS_S

  Represents a message status.

Typedefs

- typedef struct _MSG_MESSAGE_STATUS_S MSG_MESSAGE_STATUS_S

6.8.2 Typedef Documentation

6.8.2.1 typedef struct _MSG_MESSAGE_STATUS_S MSG_MESSAGE_STATUS_S

6.9 Msg_Mapi_QueryRule.h File Reference

6.9.1 Detailed Description

Defines the query related structure.

Query rule is used to query some specific messages from storage. Query rule list is composed of various query rules. Once the message satisfies one of the query rules in the list, it satisfies the query rule list.

Applications should assemble a query rule list to filter the message even there is only one query rule. Please refer to MSG_QUERY_LIST_S

Framework storage plugin SHOULD implement the query function and provide the supported queryId and queryValue for user.

```
#include "Msg_Mapi_Types.h"
```

Data Structures

- struct _MSG_QUERY_ID_LIST_S

  Represents a query Id list.

- struct _MSG_QUERY_S

  Represents a query rule.

- struct _MSG_QUERY_LIST_S

  Represents a query list.

- struct _MSG_SORT_TYPE_LIST_S

  Represents a sort type list.

- struct _MSG_SORT_RULE_S

  Represents a sort rule.

Typedefs

- typedef int MSG_QUERY_ID_T

  The query id.

- typedef struct _MSG_QUERY_ID_LIST_S MSG_QUERY_ID_LIST_S
- typedef struct _MSG_QUERY_S MSG_QUERY_S
- typedef struct _MSG_QUERY_LIST_S MSG_QUERY_LIST_S
- typedef int MSG_SORT_TYPE_T

  Represents a sort type.

- typedef struct _MSG_SORT_TYPE_LIST_S MSG_SORT_TYPE_LIST_S
- typedef struct _MSG_SORT_RULE_S MSG_SORT_RULE_S

Enumerations

- enum _MSG_QUERY_ID_E {

  MSG_QUERY_BY_UNKNOWN = 0, MSG_QUERY_BY_BEFORE_DATETIME, MSG_QUERY_BY_AFTER_DATETIME, MSG_QUERY_BY_FROM,

  MSG_QUERY_BY_TO, MSG_QUERY_BY_SUBJECT, MSG_QUERY_BY_SIZE, MSG_QUERY_BY_STATUS,

  MSG_QUERY_BY_DELIVERY_STATUS, MSG_QUERY_BY_HASATTACHMENT, MSG_QUERY_BY_HASDRMCONTENT, MSG_QUERY_BY_PRIORITY,

  MSG_QUERY_BY_USER_DEF_BEGIN = 0x00001000 }

  Represents the values of query Id.

- enum _MSG_SORT_TYPE_E {

  MSG_SORT_BY_UNKNOWN = 0, MSG_SORT_BY_DISPLAY_SUBJECT, MSG_SORT_BY_DISPLAY_FROM, MSG_SORT_BY_DISPLAY_TO,

  MSG_SORT_BY_DISPLAY_TIME, MSG_SORT_BY_SIZE, MSG_SORT_BY_STATUS, MSG_SORT_BY_DELIVERY_STATUS,

  MSG_SORT_BY_HASATTACHMENT, MSG_SORT_BY_HASDRMCONTENT, MSG_SORT_BY_PRIORITY, MSG_SORT_BY_USER_DEF_BEGIN = 0x00001000 }

  Represents the values of sort types.

6.9.2 Typedef Documentation

6.9.2.1 typedef struct _MSG_QUERY_ID_LIST_S MSG_QUERY_ID_LIST_S

6.9.2.2 typedef int MSG_QUERY_ID_T

The query id.

One query id presents a unique query type.

6.9.2.3 typedef struct _MSG_QUERY_LIST_S MSG_QUERY_LIST_S

6.9.2.4 typedef struct _MSG_QUERY_S MSG_QUERY_S

6.9.2.5 typedef struct _MSG_SORT_RULE_S MSG_SORT_RULE_S

6.9.2.6 typedef struct _MSG_SORT_TYPE_LIST_S MSG_SORT_TYPE_LIST_S

6.9.2.7 typedef int MSG_SORT_TYPE_T

Represents a sort type.

The values for this type SHOULD be in _MSG_SORT_TYPE_E

6.9.3 Enumeration Type Documentation

6.9.3.1 enum _MSG_QUERY_ID_E

Represents the values of query Id.

This enum is used as the value of MSG_QUERY_ID_T, which is used as an input parameter in method MsgQueryMessage. Framework storage plugin can extend the supported query Id in this enum. Applications can use the query Id when querying messages.

Enumerator:

MSG_QUERY_BY_UNKNOWN Unknown query Id

MSG_QUERY_BY_BEFORE_DATETIME Query message whose time is before or equal to the datetime.

MSG_QUERY_BY_AFTER_DATETIME Query message whose time is after or equal to the datetime

MSG_QUERY_BY_FROM Query message by from_keyword

MSG_QUERY_BY_TO Query message by to_keyword

MSG_QUERY_BY_SUBJECT Query message by subject_keyword

MSG_QUERY_BY_SIZE Query message by size

MSG_QUERY_BY_STATUS Query message by status

MSG_QUERY_BY_DELIVERY_STATUS Query message by delivery status

MSG_QUERY_BY_HASATTACHMENT Query message by has attachment

MSG_QUERY_BY_HASDRMCONTENT Query message by drm content

MSG_QUERY_BY_PRIORITY Query message by priority

MSG_QUERY_BY_USER_DEF_BEGIN User defined query Id begin value

6.9.3.2 enum _MSG_SORT_TYPE_E

Represents the values of sort types.

This enum is used as the value of MSG_SORT_TYPE_T.

Enumerator:

MSG_SORT_BY_UNKNOWN Unknown sort type

MSG_SORT_BY_DISPLAY_SUBJECT Sort by display subject

MSG_SORT_BY_DISPLAY_FROM Sort by display from

MSG_SORT_BY_DISPLAY_TO Sort by display to

MSG_SORT_BY_DISPLAY_TIME Sort by display time

MSG_SORT_BY_SIZE Sort by size

MSG_SORT_BY_STATUS Sort by status

MSG_SORT_BY_DELIVERY_STATUS Sort by delivery status

MSG_SORT_BY_HASATTACHMENT Sort by has attachment

MSG_SORT_BY_HASDRMCONTENT Sort by drm content

MSG_SORT_BY_PRIORITY Sort by priority

MSG_SORT_BY_USER_DEF_BEGIN User defined sort type begin value

6.10 Msg_Mapi_Session.h File Reference

6.10.1 Detailed Description

Defines session handling and message sending and receiving related functions.

```
#include "Msg_Mapi_Message.h"
#include "Msg_Mapi_Message_Status.h"
#include "Msg_Mapi_Message_Request.h"
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_FilterRule.h"
```

Typedefs

- typedef struct MessagingSession * MSG_SESSION_HANDLE_T

  Defines a session handle.

- typedef void(* MsgOnMessageIncomingCallback)(MSG_MESSAGE_S *pMsgInfo, MSG_MESSAGE_STATUS_S *pMsgStatus, MSG_SESSION_HANDLE_T hSession)

  Prototype of the function that will be called when a new message received.

- typedef void(* MsgOnStatusChangedCallback)(MSG_MESSAGE_STATUS_S *pMsgStatus, MSG_SESSION_HANDLE_T hSession)

  Prototype of the function that will be called when status changed.

Functions

- MSG_ERROR_T MsgCreateSession (MSG_SESSION_HANDLE_T *hSession)

  Creates a session handle.

- MSG_ERROR_T MsgDestroySession (MSG_SESSION_HANDLE_T hSession)

  Destroys the session handle.

- MSG_ERROR_T MsgRegMessageCallback (MSG_SESSION_HANDLE_T hSession, MsgOnMessageIncomingCallback onMsgIncoming)

  Registers MsgOnMessageIncomingCallback function to the session handle.

- MSG_ERROR_T MsgRegStatusCallback (MSG_SESSION_HANDLE_T hSession, MsgOnStatusChangedCallback OnStatusChanged)

  Registers MsgOnStatusChangedCallback to the session handle.

- MSG_ERROR_T MsgConnectSession (MSG_SESSION_HANDLE_T hSession)

  Connects the session handle.

- MSG_ERROR_T MsgDisconnectSession (MSG_SESSION_HANDLE_T hSession)

  Disconnects the session handle.

- MSG_ERROR_T MsgSetPriority (MSG_SESSION_HANDLE_T hSession, MSG_SESSION_-PRIORITY_T priority)

  Sets priority for the session handle.

- MSG_ERROR_T MsgSetFilterList (MSG_SESSION_HANDLE_T hSession, const MSG_-FILTER_LIST_S *pFilterList)

  Sets filter list to the session handle.

- MSG_ERROR_T MsgSubmitReq (MSG_SESSION_HANDLE_T hSession, const MSG_-MESSAGE_REQUEST_S *pReq, MSG_REQUEST_ID_T *pReqId)

  Submits a request.

- MSG_ERROR_T MsgCancelReq (MSG_SESSION_HANDLE_T hSession, MSG_REQUEST_-ID_T reqId)

  Cancels a request by request id.

- MSG_ERROR_T MsgGetAllSupportedMsgType (MSG_MESSAGE_TYPE_LIST_S *pMsgTypeList)

  Gets all supported message types.

- void MsgReleaseMsgTypeList (MSG_MESSAGE_TYPE_LIST_S *pMsgTypeList)

  Releases the memory of message type List.

- MSG_ERROR_T MsgGetAllSupportedFilterId (MSG_FILTER_ID_LIST_S *pFilterIdList)

  Gets all supported filter id.

- void MsgReleaseFilterIdList (MSG_FILTER_ID_LIST_S *pFilterIdList)

  Releases the memory of filter Id List.

- MSG_PLUGIN_ERROR_T MsgGetSessionPluginError (MSG_SESSION_HANDLE_T hSes-sion)

  Get session specific plugin error.

6.10.2 Typedef Documentation

6.10.2.1 typedef struct MessagingSession* MSG_SESSION_HANDLE_T

Defines a session handle.

For details of Session, please refer to Session

6.10.2.2 typedef void(* MsgOnMessageIncomingCallback)(MSG_MESSAGE_S *pMsgInfo, MSG_MESSAGE_STATUS_S *pMsgStatus, MSG_SESSION_HANDLE_T hSession)

Prototype of the function that will be called when a new message received.

Applications SHOULD implement this callback function and register it into a session handle. For how to register this callback function, please refer to MsgRegMessageCallback. This callback function will be called when a new message received and the message satisfies the filter list which is attached to the session handle. The callback function runs in application process, not in framework process. The memory

for `pMsgInfo` and `pMsgStatus` are managed by MAPI, when the callback function is finished, the memory will be released by MAPI automatically.

Parameters:

← ***pMsgInfo*** is a pointer to the received message.

← ***pMsgStatus*** is a pointer to the status.

← ***hSession*** is a session handle. It indicates which session is invoked.

Returns:

void

Please refer to Callback Functions

Since:

1.0

6.10.2.3 typedef void(* MsgOnStatusChangedCallback)(MSG_MESSAGE_STATUS_S *pMsgStatus, MSG_SESSION_HANDLE_T hSession)

Prototype of the function that will be called when status changed.

Applications SHOULD to implement this callback function and register it into a session handle. For how to register this callback function, please refer to MsgRegStatusCallback. For instance, if application sends a message, this callback function will be called to report sending status. Application can get the request id from pMsgStatus to know which request the status is bound. The callback function runs in user application process, not in framework process. The memory of `pMsgStatus` is managed by MAPI, when the callback function is finished, the memory will be released by MAPI automatically.

Parameters:

← ***pMsgStatus*** is a pointer to the status.

← ***hSession*** is a session handle. It indicates which session is invoked.

Returns:

void

Note:

Please refer to Callback Functions

Since:

1.0

6.10.3 Function Documentation

6.10.3.1 MSG_ERROR_T MsgCancelReq (MSG_SESSION_HANDLE_T *hSession*, MSG_REQUEST_ID_T *reqId*)

Cancels a request by request id.

If cancel successfully, status "MSG_STATUS_REQ_CANCELLED" will be sent back. If cancel failed, no special status will be sent back.

If reqId is invalid, MSG_ERR_INVALID_REQUEST_ID will be returned.

This is an asynchronous function.

Parameters:

← ***hSession*** is a pointer to an established session handle.

← ***reqId*** is the request id of the request to be cancelled.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS submits the "cancel request" request to framework successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_SESSION_NOT_CONNECTED session is not connected.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_INVALID_REQUEST_ID request id is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetSessionPluginError() to get detail error code.

Note:

The result information will be sent back by using callback function MsgOnStatusChangedCallback

Since:

1.0

6.10.3.2 MSG_ERROR_T MsgConnectSession (MSG_SESSION_HANDLE_T *hSession*)

Connects the session handle.

This function MUST be called after session handle is created.

It is the precondition to call functions MsgSubmitReq, MsgSetPriority, MsgSetFilterList, MsgCancelReq and MsgGetAllSupportedFilterId

Parameters:

← ***hSession*** is a pointer to an session handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS connects the session handle successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.3 MSG_ERROR_T MsgCreateSession (MSG_SESSION_HANDLE_T * *hSession*)

Creates a session handle.

This function creates a session handle. Each application can call this function multiple times to create more than one session handles. All session handles are independent. For details of session, please refer to Session.

Parameters:

→ ***hSession*** is a pointer to a session handle. When this function returns MSG_SUCCESS, `*hSession` will be valid session handle. Otherwise, it will be `NULL`.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS creates session handle successfully.

MSG_ERR_NULL_POINTER the input parameter is NULL.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.4 MSG_ERROR_T MsgDestroySession (MSG_SESSION_HANDLE_T *hSession*)

Destroys the session handle.

This function destroys a session handle. This function releases resources and disconnects session if it's connected. Once a session handle is destroyed, it can't be used any more. For each created session, applications MUST destroy it, when it's not used any more. If the input parameter is NULL, this function does nothing and returns MSG_ERR_INVALID_SESSION_HANDLE.

Parameters:

← ***hSession*** is a pointer to an session handle to be destroyed.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS destroyes the handle successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

Since:

1.0

6.10.3.5 MSG_ERROR_T MsgDisconnectSession (MSG_SESSION_HANDLE_T *hSession*)

Disconnects the session handle.

If the session is NULL, this function does nothing and returns MSG_ERR_INVALID_SESSION_-HANDLE.

Parameters:

← ***hSession*** is a pointer to an session handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS disconnects the session handle successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_SESSION_NOT_CONNECTED the session handle is not connected.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.6 MSG_ERROR_T MsgGetAllSupportedFilterId (MSG_FILTER_ID_LIST_S * *pFilterIdList*)

Gets all supported filter id.

Parameters:

→ ***pFilterIdList*** is a pointer to a filter id list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS gets all supported filter id successfully.

MSG_ERR_NULL_POINTER pFilterIdList is NULL.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error

6.10.3.7 MSG_ERROR_T MsgGetAllSupportedMsgType (MSG_MESSAGE_TYPE_LIST_S * *pMsgTypeList*)

Gets all supported message types.

Parameters:

→ ***pMsgTypeList*** is a pointer to a message type list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS gets all supported message types successfully.

MSG_ERR_NULL_POINTER pMsgTypeList is NULL.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error

6.10.3.8 MSG_PLUGIN_ERROR_T MsgGetSessionPluginError (MSG_SESSION_HANDLE_T *hSession*)

Get session specific plugin error.

This function is to get the specific plugin error. In case that some functions return MSG_ERR_PLUGIN, applications can call this function immediately to get specific error code of the plugins. If hSession is invalid, this function will return 0.

Parameters:

← ***hSession*** is a pointer to an established session handle.

Returns:

MSG_PLUGIN_ERROR_T

6.10.3.9 MSG_ERROR_T MsgRegMessageCallback (MSG_SESSION_HANDLE_T *hSession*, MsgOnMessageIncomingCallback *onMsgIncoming*)

Registers MsgOnMessageIncomingCallback function to the session handle.

Applications SHOULD implement the function MsgOnMessageIncomingCallback. For details, please refer to MsgOnMessageIncomingCallback

Applications MAY deregister the callback function MsgOnMessageIncomingCallback by invoking this function like `MsgRegMessageCallback(hSession, NULL)`.

This function MUST be called after session handle is created. It makes no difference to call this function before or after session connected. This function can be called more times and the last inovking will be effective.

Parameters:

← ***hSession*** is a pointer to an session handle.

← ***onMsgIncoming*** is a function pointer to an implemenation function.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS registers the call back function successfully

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.10 MSG_ERROR_T MsgRegStatusCallback (MSG_SESSION_HANDLE_T *hSession*, MsgOnStatusChangedCallback *OnStatusChanged*)

Registers MsgOnStatusChangedCallback to the session handle.

Applications SHOULD implement the function MsgOnStatusChangedCallback. For details, please refer to MsgOnStatusChangedCallback

Applications MAY deregister the callback function MsgOnStatusChangedCallback by invoking this function like `MsgRegStatusCallback(hSession, NULL)`.

This function MUST be called after session handle is created. It makes no difference to call this function before or after session connected. This function can be called more times and the last invoking will be effetive.

Parameters:

← ***hSession*** is a pointer to an session handle.

← ***OnStatusChanged*** is a function pointer to an implemenation function.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS registers the call back function successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.11 void MsgReleaseFilterIdList (MSG_FILTER_ID_LIST_S * *pFilterIdList*)

Releases the memory of filter Id List.

This function releases the memory of filterIds in MSG_FILTER_ID_LIST_S structure.

If pFilterIdList or pFilterIdList->filterIds is NULL, nothing happens. If pFilterIdList or pFilterIdList->filterIds is invalid, undefined behavior.

Note:

This function only frees the list pointer member inside the structure. It doesn't free the structure itself.

Parameters:

← ***pFilterIdList*** is a pointer to an MSG_FILTER_ID_LIST_S structure.

Since:

1.0

6.10.3.12 void MsgReleaseMsgTypeList (MSG_MESSAGE_TYPE_LIST_S * *pMsgTypeList*)

Releases the memory of message type List.

This function releases the memory of pMsgTypes in MSG_MESSAGE_TYPE_LIST_S structure. If pMsgTypeList or pMsgTypeList->pMsgTypes is NULL, nothing happens. If pMsgTypeList or pMsgTypeList->pMsgTypes is invalid, undefined behavior.

Note:

This function only frees the list pointer member inside the structure. It doesn't free the structure itself.

Parameters:

← ***pMsgTypeList*** is a pointer to a MSG_MESSAGE_TYPE_LIST_S structure.

Since:

1.0

6.10.3.13 MSG_ERROR_T MsgSetFilterList (MSG_SESSION_HANDLE_T *hSession*, const MSG_FILTER_LIST_S * *pFilterList*)

Sets filter list to the session handle.

This function sets the filter list to a session handle. It impacts the callback functions. Only the received messages satisfying one filter in the filter list can be passed up to the callback function. Otherwise, the callback function will not be called. For details of filter list, please refer to MSG_FILTER_LIST_S

This function MUST be called after session is created and connected. This function can be called more times and the last one will be effective. If no filter list is set to the session or pFilterList is NULL, all the messages will be received by the session.

Parameters:

← ***hSession*** is a pointer to an established session handle.

← ***pFilterList*** is a pointer to the filter list for the session handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS sets the filter list successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_SESSION_NOT_CONNECTED the session handle is not connected.

MSG_ERR_INVALID_FILTER_ID the filter id in pFilterList is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.14 MSG_ERROR_T MsgSetPriority (MSG_SESSION_HANDLE_T *hSession*, MSG_SESSION_PRIORITY_T *priority*)

Sets priority for the session handle.

This function sets the priority for a session handle. The following use case shows the usage of priority:

When a messaging is received, if this message satisfies the filter rule of this session and intercept of the satisfied filter rule is set true, this message will be passed to this session callback function and will not be passed to lower priority session callback functions.

All the sessions will receive the messages if they have same priority.

For details of intercept, please refer to Intercept.

This function MUST be called after session is created and connected. This function can be called more times and the last invoking will be effective.

If this function is not called, the default priority is MSG_SESSION_PRIORITY_NORMAL.

Parameters:

← ***hSession*** is a pointer to a session handle.

← ***priority*** is for the session handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS sets the priority successfully.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_SESSION_NOT_CONNECTED the session handle is not connected.

MSG_ERR_INVALID_SESSION_PRIORITY the priority is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.10.3.15 MSG_ERROR_T MsgSubmitReq (MSG_SESSION_HANDLE_T *hSession*, const MSG_MESSAGE_REQUEST_S * *pReq*, MSG_REQUEST_ID_T * *pReqId*)

Submits a request.

To submit a request, applications need to specify the reqType in request structure: For example, to send a message, applications need to set `reqType` with value MSG_REQUEST_TYPE_SEND.

`reqId` will be filled by this function automatically and return to application.

Following is the sample code to send a message:

```
MSG_SESSION_HANDLE_T session;
MSG_MESSAGE_REQUEST_S req;
MSG_MESSAGE_S msg;
//.... init session and msg.
//Set request type
req.reqType = MSG_REQUEST_TYPE_SEND;
//Set msg
req.mpMsg = &msg;
//Submit request.
MsgSubmitReq(session, &req);
```

Message plugins can extend the request types to provide more functions. Please refer to enum MSG_-REQUEST_TYPE_T to define the extended request types. When applications submit these requests, the framework will pass the requests to the underlaying plugins.

If pReq->reqType is invalid, an asynchronous status with error "MSG_ERR_INVALID_REQUEST" will be sent back.

For request type "MSG_REQUEST_TYPE_SEND" and "MSG_REQUEST_TYPE_SENDSAVE":

Applications MUST fill valid message type for field pReq->msgType, otherwise, MSG_ERR_INVALID_-REQUEST will be returned immediately.

- If pReq->pMsg->accountId is MSG_DEFAULT_ACCOUNT_ID, then the default account is used.
- If pReq->pMsg->accountId is not valid, the function returns "MSG_ERR_INVALID_REQUEST".

For request type "MSG_REQUEST_TYPE_SEND", following fields will be ignored:

pReq->pMsg->msgId, pReq->pMsg->parentId, pReq->pMsg->storageId.

For request type "MSG_REQUEST_TYPE_SENDSAVE", if valid pReq->pMsg->parentId and pReq->pMsg->storageId are given, they override the default values in account information. If pReq->pMsg->parentId is MSG_DEFAULT_FOLDER_ID, the message will be added to the outbox in its associated account. If pReq->pMsg->storageId is MSG_STORAGE_DEFAULT, the message will be added to the `defaultMsgStorageId` in its associated account.

This is an asynchronous function.

Parameters:

← ***hSession*** is a pointer to an established session handle.

← ***pReq*** is the request to be submitted.

→ ***pReqId*** is the id of this request. This id is assigned by framework automatically. Applications can keep this id and use MsgCancelReq to cancel the request.

Returns:

MSG_ERROR_T.

Return values:

MSG_SUCCESS submits the request to framework successfully.

MSG_ERR_NULL_POINTER pReq is NULL or pReqId is NULL.

MSG_ERR_INVALID_SESSION_HANDLE the session handle is invalid.

MSG_ERR_SESSION_NOT_CONNECTED session is not connected.

MSG_ERR_INVALID_REQUEST the message type in request is invalid. For request type "MSG_-REQUEST_TYPE_SENDSAVE", some mandatory fields are invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetSessionPluginError() to get detail error code.

Note:

The result information will be sent back by using callback function MsgOnStatusChangedCallback.

Since:

1.0

6.11 Msg_Mapi_Storage_Account.h File Reference

6.11.1 Detailed Description

Defines account related functions.

This file includes all MAPI account functions, such as creating, updating, querying, and deleting account.

There are also some functions used to set and get default account id.

```
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Account.h"
#include "Msg_Mapi_Storage_Manager.h"
```

Functions

- MSG_ERROR_T MsgAddAccount (MSG_CONNECTION_HANDLE_T hConnection, MSG_ACCOUNT_S *pAccount)

  Adds a new account.

- MSG_ERROR_T MsgUpdateAccount (MSG_CONNECTION_HANDLE_T hConnection, const MSG_ACCOUNT_S *pAccount)

  Updates an account.

- MSG_ERROR_T MsgDeleteAccount (MSG_CONNECTION_HANDLE_T hConnection, MSG_ACCOUNT_ID_T accountId, bool delRootFolder)

  Deletes an account.

- MSG_ERROR_T MsgCountAccount (MSG_CONNECTION_HANDLE_T hConnection, const MSG_MESSAGE_TYPE_S *pMsgType, int *pCount)

  Gets the number of accounts for the specified message type.

- MSG_ERROR_T MsgQueryAccount (MSG_CONNECTION_HANDLE_T hConnection, const MSG_MESSAGE_TYPE_S *pMsgType, MSG_ACCOUNT_ID_LIST_S *pAccountIdList)

  Queries account id list for the specified message type.

- MSG_ERROR_T MsgGetAccount (MSG_CONNECTION_HANDLE_T hConnection, MSG_ACCOUNT_ID_T accountId, MSG_ACCOUNT_S **ppAccount)

  Gets the detailed account information for the specified account.

- void MsgReleaseAccount (MSG_ACCOUNT_S *pAccount)

  Releases the memory of an account structure.

- MSG_ERROR_T MsgGetCommAccount (MSG_CONNECTION_HANDLE_T hConnection, MSG_ACCOUNT_ID_T accountId, MSG_ACCOUNT_S **ppAccount)

  Gets account common information.

- void MsgReleaseCommAccount (MSG_ACCOUNT_S *pAccount)

  Releases account common information.

- MSG_ERROR_T MsgGetDefaultAccountID (MSG_CONNECTION_HANDLE_T hConnection, const MSG_MESSAGE_TYPE_S *pMsgType, MSG_ACCOUNT_ID_T *pAccountId)

  Gets the default account id.

- MSG_ERROR_T MsgSetDefaultAccountID (MSG_CONNECTION_HANDLE_T hConnection, const MSG_MESSAGE_TYPE_S *pMsgType, MSG_ACCOUNT_ID_T accountId)

  Sets the default account id.

- void MsgReleaseAccountIdList (MSG_ACCOUNT_ID_LIST_S *pAccountIdList)

  Release the memory of an account id list.

- MSG_ERROR_T MsgGetAccountIdsByStorage (MSG_CONNECTION_HANDLE_T hConnection, MSG_STORAGE_ID_T storageId, MSG_ACCOUNT_ID_LIST_S *pAccountIdList)

  Gets all the account ids on a specified storage.

- MSG_ERROR_T MsgMoveAccountToStorage (MSG_CONNECTION_HANDLE_T hConnection, MSG_ACCOUNT_ID_T accountId, MSG_STORAGE_ID_T storageId)

  Moves an account to the specified storage.

6.11.2 Function Documentation

6.11.2.1 MSG_ERROR_T MsgAddAccount (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_ACCOUNT_S * *pAccount*)

Adds a new account.

This function adds an account into storage. Application fills all the account information into the account structure `pAccount` except for field `accountId`. The `accountId` SHOULD have no value, and after the account is successfully created, it will be filled with the new generated account id. Mandatory fields of account structure MUST be valid, otherwise the function will be failed. The mandatory fields for adding operation are `msgType`, `rootFolderId`, `storageId`, `defaultMsgStorageId`.

Parameters:

← ***hConnection*** is a connection handle.

↔ ***pAccount*** is the data user defined for this new account.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pAccount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_ACCOUNT The mandatory fields of account structure are invalid.

MSG_ERR_STORAGE_FULL Storage is FULL.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_MAX_NUMBER_REACHED max number reached.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

For the sample code, please refer to Account Storage: Creating and Storing an Account.

Since:

1.0

6.11.2.2 MSG_ERROR_T MsgCountAccount (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_MESSAGE_TYPE_S * *pMsgType*, int * *pCount*)

Gets the number of accounts for the specified message type.

This function gets the number of accounts for the specified message type, such as SMS, MMS, and Email account. If the `pMsgType` is NULL, the function will return the number of all accounts. If the `pMsgType` is a specific type, it will return the number of accounts of that type. If the `pMsgType` can't match with anyone above, the function is failed.

Parameters:

← ***hConnection*** is a connection handle.

← ***pMsgType*** is a pointer to the specified message type.

→ ***pCount*** is a pointer to the account number.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE The message type is not valid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.3 MSG_ERROR_T MsgDeleteAccount (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_ACCOUNT_ID_T *accountId*, bool *delRootFolder*)

Deletes an account.

This function deletes an account by account id.

If delRootFolder is FALSE, the function will only delete the account and will not delete the root folder. In this case, MSG_SUCCESS will be returned.

If delRootFolder is TRUE and the root folder of this account is not associated with other account(s), this function will delete both account and the root folder(including all the content inside). If the root folder is associated with other account(s), this function will only delete the account and will not delete the root folder. In this case, MSG_ERR_ROOTFOLDER_INUSE will be returned.

Parameters:

← ***hConnection*** is a connection handle.

← ***accountId*** indicates the specific account need to be deleted.

← ***delRootFolder*** indicates whether delete the associated root folder and the content inside.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_ACCOUNT_ID This account is not exist.

MSG_ERR_ROOTFOLDER_INUSE This account is deleted, but the root folder is not deleted.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.4 MSG_ERROR_T MsgGetAccount (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_ACCOUNT_ID_T *accountId*, MSG_ACCOUNT_S ** *ppAccount*)

Gets the detailed account information for the specified account.

This function gets all the detailed account information for the specified account, including common and extended data. The memory of out parameter `ppAccount` is allocated in this function. Application SHOULD use function MsgReleaseAccount to release its memory explicitly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***accountId*** identifies the queried account.

→ ***ppAccount*** is a pointer to a pointer to the account structure which contains all this account detail information.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `ppAccount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_ACCOUNT_ID This account is not exist.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.5 MSG_ERROR_T MsgGetAccountIdsByStorage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_STORAGE_ID_T *storageId*, MSG_ACCOUNT_ID_LIST_S * *pAccountIdList*)

Gets all the account ids on a specified storage.

This function gets all account ids on a specified storage.The parameter `pAccountIdList` is defined outside this funtion, but the memory of its field `accountIds` is allocated in this function. The function caller SHOULD use function MsgReleaseAccountIdList to release `accountIds` memory explicitlly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***storageId*** indicates the queried storage.

→ ***pAccountIdList*** is a pointer to the account id list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pAccountIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_STORAGE Storage id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.6 MSG_ERROR_T MsgGetCommAccount (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_ACCOUNT_ID_T *accountId*, MSG_ACCOUNT_S ** *ppAccount*)

Gets account common information.

This function gets the account common information. The account common information includes all fields of structure MSG_ACCOUNT_S except for `pExData`. The memory block of out parameter `ppAccount` is allocated in this function. Application SHOULD use function MsgReleaseCommAccount to release its memory explicitly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***accountId*** iendtifies the queried account.

→ ***ppAccount*** is a pointer to a pointer to the structure which contains all the account common information.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `ppAccount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_ACCOUNT_ID This account is not exist.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.7 MSG_ERROR_T MsgGetDefaultAccountID (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_MESSAGE_TYPE_S * *pMsgType*, MSG_ACCOUNT_ID_T * *pAccountId*)

Gets the default account id.

This function gets the default account id for the specified message type. The parameter `pMsgType` cannot be NULL, for each message type has one default account.

Parameters:

← ***hConnection*** is a connection handle.

← ***pMsgType*** is a pointer to the specified message type.

→ ***pAccountId*** is a pointer to the default account id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsgType` or `pAccountId` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE The message type is not valid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.8 MSG_ERROR_T MsgMoveAccountToStorage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_ACCOUNT_ID_T *accountId*, MSG_STORAGE_ID_T *storageId*)

Moves an account to the specified storage.

This function moves an account from a storage to the specified storage.

Parameters:

← ***hConnection*** is a connection handle.

← ***storageId*** indicates the destination storage.

← ***accountId*** indicates the moved account.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_STORAGE Storage id is invalid.

MSG_ERR_INVALID_ACCOUNT_ID This account is not exist.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_MAX_NUMBER_REACHED max number reached.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.9 MSG_ERROR_T MsgQueryAccount (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_MESSAGE_TYPE_S * *pMsgType*, MSG_ACCOUNT_ID_LIST_S * *pAccountIdList*)

Queries account id list for the specified message type.

This function gets account id list for the specified message type. The parameter `pAccountIdList` memory is allocated by the function caller, but the memory of its field `accountIds` is allocated in this function, Application SHOULD use function MsgReleaseAccountIdList to release the memory explicitly. However if this function is failed, the memory is NOT allocated in this function.

If the `pMsgType` is NULL, this function will return all account id list. If the `pMsgType` is a specific type, this function will return account id list of that type. If the `pMsgType` cann't match with anyone above, this function is failed.

Parameters:

← ***hConnection*** is a connection handle.

← ***pMsgType*** is a pointer to the specified message type.

→ ***pAccountIdList*** is a pointer to the list of account id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pAccountIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE The message type is not valid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.11.2.10 void MsgReleaseAccount (MSG_ACCOUNT_S * *pAccount*)

Releases the memory of an account structure.

This function releases the memory of an account structure. If pAccount is NULL, nothing happens. If pAccount is invalid, undefined behavior.

Parameters:

← ***pAccount*** is a pointer to an account structure.

Since:

1.0

6.11.2.11 void MsgReleaseAccountIdList (MSG_ACCOUNT_ID_LIST_S * *pAccountIdList*)

Release the memory of an account id list.

This function frees the memory of pointer `accountIds` inside MSG_ACCOUNT_ID_LIST_S. If pAccountIdList or pAccountIdList->accountIds is NULL, nothing happens. If pAccountIdList or pAccountIdList->accountIds is invalid, undefined behavior.

Note:

This function does NOT free this structure itself.

Parameters:

← ***pAccountIdList*** points to a account id list.

Since:

1.0

6.11.2.12 void MsgReleaseCommAccount (MSG_ACCOUNT_S * *pAccount*)

Releases account common information.

This function releases the memory of account common information , which is allocated in function MsgGetCommAccount. The account common information includes all the members of structure MSG_-ACCOUNT_S except for `pExData`. If pAccount is NULL, nothing happens. If pAccount is invalid, undefined behavior.

Parameters:

← ***pAccount*** is a pointer to an account structure which only contains common information.

Since:

1.0

6.11.2.13 MSG_ERROR_T MsgSetDefaultAccountID (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_MESSAGE_TYPE_S * *pMsgType*, MSG_ACCOUNT_ID_T *accountId*)

Sets the default account id.

This function sets the default account id for the specified message type. The parameter `pMsgType` cannot be NULL, for each message type has one default account.

Parameters:

← ***hConnection*** is a connection handle.

← ***pMsgType*** is a pointer to the specified message type.

← ***accountId*** is the account id which is to be set as the default account.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsgType` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE The message type is not valid.

MSG_ERR_INVALID_ACCOUNT_ID This account is not exist.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Example code Account Storage: Getting a Default Account

Since:

1.0

6.11.2.14 MSG_ERROR_T MsgUpdateAccount (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_ACCOUNT_S * *pAccount*)

Updates an account.

The function is expected to update msgType, rootFolderId, and defaultMsgStorageID for the account indentified by the given accountId as long as the given values are valid. Optionally, the function updates extension data given by pExtData and extDataSize, and accountName, if necessary.

All the data fields, except for`accountId` and `storageId`, can be updated.

Parameters:

← ***hConnection*** is a connection handle.

← ***pAccount*** is the data user defined to update the account

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pAccount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_ACCOUNT_ID This account is not exist.

MSG_ERR_INVALID_ACCOUNT The mandatory fields except accountId are invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

For the sample code, please refer to Account Storage: Updating an Account

Since:

1.0

6.12 Msg_Mapi_Storage_Folder.h File Reference

6.12.1 Detailed Description

Defines folder related functions in MAPI.

This file includes all folder functions, such as creating, updating, queryring, deleting folder. For the overview on folder, please refer to Folder .

```
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Folder.h"
#include "Msg_Mapi_Storage_Manager.h"
```

Functions

- MSG_ERROR_T MsgAddFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_S *pFolder)

  Adds a new folder.

- MSG_ERROR_T MsgCreateRootFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_S *pFolder)

  Creates a new root folder.

- MSG_ERROR_T MsgUpdateFolder (MSG_CONNECTION_HANDLE_T hConnection, const MSG_FOLDER_S *pFolder)

  Updates a folder.

- MSG_ERROR_T MsgDeleteFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, bool recursive)

  Deletes a folder.

- MSG_ERROR_T MsgCountSubFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, int *pCount)

  Gets the number of sub folders under the specified folder.

- MSG_ERROR_T MsgCountSubFolderRecursive (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, int *pCount)

  Gets the number of sub folders under the specified folder.

- MSG_ERROR_T MsgQueryFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, int iBeginPos, int iCount, MSG_FOLDER_ID_LIST_S *pFolderIdList)

  Queries sub folder ids under the specified folder.

- MSG_ERROR_T MsgQueryFolderRecursive (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, int iBeginPos, int iCount, MSG_FOLDER_ID_LIST_S *pFolderIdList)

  Recursively queries sub folder ids under the specified folder.

- MSG_ERROR_T MsgMoveFolderToFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, MSG_FOLDER_ID_T destFolderId)

Moves a folder to the specified folder.

- MSG_ERROR_T MsgMoveFolderToStorage (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, MSG_STORAGE_ID_T destStorageId)

  Moves a folder to another storage.

- MSG_ERROR_T MsgGetFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, MSG_FOLDER_S **ppFolderInfo)

  Gets the folder information of the specified folder.

- MSG_ERROR_T MsgGetFolderByType (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T rootFolderId, MSG_FOLDER_TYPE_T folderType, MSG_FOLDER_ID_LIST_S *pFolderIdList)

  Gets the folder by folder type for the specified root folder.

- MSG_ERROR_T MsgGetPredefinedFolders (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T rootFolderId, MSG_FOLDER_S **ppInbox, MSG_FOLDER_S **ppOutbox, MSG_FOLDER_S **ppDraft, MSG_FOLDER_S **ppSent, MSG_FOLDER_S **ppDeleted)

  Gets the predefined folders for the specified root folder.

- MSG_ERROR_T MsgGetFolderParentId (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId, MSG_FOLDER_ID_T *pParentFolderId)

  Gets the parent folder id of a folder.

- void MsgReleaseFolder (MSG_FOLDER_S *pFolderInfo)

  Releases a memory block of a folder structure.

- void MsgReleaseFolderIdList (MSG_FOLDER_ID_LIST_S *pFolderIdList)

  Releases a memory block of folder id list.

- MSG_ERROR_T MsgGetAllRootFolderIds (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_LIST_S *pFolderIdList)

  Get all root folders.

- MSG_ERROR_T MsgGetFolderIdsByStorage (MSG_CONNECTION_HANDLE_T hConnection, MSG_STORAGE_ID_T storageId, MSG_FOLDER_ID_LIST_S *pFolderIdList)

  Get all the folder id list on a specified storage.

6.12.2 Function Documentation

6.12.2.1 MSG_ERROR_T MsgAddFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_S * *pFolder*)

Adds a new folder.

This function creates a folder under the specified parent folder. This function can only create user defined folder, and not work for root and predefined folder. Mandatory fields of folder structure MUST be valid, otherwise the function will be failed. The mandatory fields for adding operation are `parentId`, `storageId`, `folderType`. If the sortType is invalid, the default value MSG_SORT_BY_UNKNOWN will be set.

Parameters:

← ***hConnection*** is a connection handle.

↔ ***pFolder*** points to the data user defined for the new folder.
pFolder->folderId will be set to a valid folder id in storage when this function is returned successfully.

Note:

pFolder->folderId = MSG_INVALID_ID [IN]
pFolder->folderId = Valid folder id in storage. [OUT]

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolder` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_TYPE Folder type is not valid for this operation.

MSG_ERR_INVALID_FOLDER The value of mandatory fields are invalid.

MSG_ERR_STORAGE_FULL Storage is FULL.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_MAX_NUMBER_REACHED max number reached.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

For sample code, please refer to:

- Folder Storage: Creating and Storing a Folder

Since:

1.0

6.12.2.2 MSG_ERROR_T MsgCountSubFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, int * *pCount*)

Gets the number of sub folders under the specified folder.

This function gets the number of sub folders, and it will not recursively search all the sub folders under the specified folder.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** identifies the specified folder.

→ ***pCount*** returns the number of the sub folders.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.3 MSG_ERROR_T MsgCountSubFolderRecursive (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, int * *pCount*)

Gets the number of sub folders under the specified folder.

This function gets the number of sub folders, and it will recursively search all the sub folders under the specified folder.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** identifies the specified folder.

→ ***pCount*** is a pointer to the number of the sub folders.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.4 MSG_ERROR_T MsgCreateRootFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_S * *pFolder*)

Creates a new root folder.

This function creates a new root folder at the application top level, and then creates five predefined folders(inbox/outbox/sent/deleted/draft) under this new root folder. It will return success when the root folder and its five predefined folders are all created successfully. Messaging application can has more than one root folders. Mandatory fields of folder structure MUST be valid, otherwise the function will be failed. The mandatory fields for adding operation are `parentId`, `storageId`, `folderType`. If the sortType is invalid, the default value MSG_SORT_BY_UNKNOWN will be set.

Parameters:

← ***hConnection*** is a connection handle.

↔ ***pFolder*** points to the data user defined for the new folder.

pFolder->folderId will be set to a valid folder id in storage when this function is returned successfully.

Note:

pFolder->folderId = MSG_INVALID_ID [IN]
pFolder->folderId = Valid folder id in storage. [OUT]

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolder` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_TYPE Folder type is invalid. (In this case, it MUST be MSG_FOLDER_TYPE_ROOT.)

MSG_ERR_INVALID_FOLDER Folder information is invalid.

MSG_ERR_STORAGE_FULL Storage is FULL.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.5 MSG_ERROR_T MsgDeleteFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, bool *recursive*)

Deletes a folder.

This function deletes a folder.

If the parameter `recursive` is false, the function will only delete this folder itself in case there is no content in this folder. If there is content in this folder, this function will return MSG_ERR_NOT_EMPTY_-FOLDER.

If the parameter `recursive` is true, regardless of whether content is in this folder, the function will recursively delete all its sub folders, messages and this folder itself.

When deleting a root folder, if the root folder is associated with account(s), the function will not delete the root folder and return MSG_ERR_ROOTFOLDER_INUSE.

Predefined folder except root folder cann't be deleted individually. Otherwise, MSG_ERR_INVALID_-FOLDER_TYPE will be returned.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** identifies the deleted folder.

← ***recursive*** indicates whether do a recursively deleting operation.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_TYPE Folder type is not valid for this operation.

MSG_ERR_NOT_EMPTY_FOLDER Folder has content, but deleting operation is not recursive.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_ROOTFOLDER_INUSE The root folder is associated with account(s) and can not be deleted.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.6 MSG_ERROR_T MsgGetAllRootFolderIds (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_LIST_S * *pFolderIdList*)

Get all root folders.

This function gets all root folder id list. The parameter `pFolderIdList` is defined outside this funtion, but the memory of its field `folderIds` is allocated in this function. The function caller should use function MsgReleaseFolderIdList to release pFolderIdList memory explicitlly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

→ ***pFolderIdList*** is a pointer to the root folder id list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolderIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.7 MSG_ERROR_T MsgGetFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, MSG_FOLDER_S ** *ppFolderInfo*)

Gets the folder information of the specified folder.

This function gets all information of the specified folder, which includes common and extended information. For this folder structure may have extended structure, the memory of folder structure is allocated in this function. Application MUST release the memory by using function MsgReleaseFolder. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** indicates the searched folder id.

→ ***ppFolderInfo*** is a pointer to a pointer to all the detail information.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `ppFolderInfo` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

For releasing folder structure memory, please refer to MsgReleaseFolder

Since:

1.0

6.12.2.8 MSG_ERROR_T MsgGetFolderByType (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *rootFolderId*, MSG_FOLDER_TYPE_T *folderType*, MSG_FOLDER_ID_LIST_S * *pFolderIdList*)

Gets the folder by folder type for the specified root folder.

This function gets the folder by folder type for the specified root folder. For example, it can retrieve inbox/outbox folders of a root folder. The parameter `pFolderIdList` is defined outside this funtion, but the memory of its field `folderIds` is allocated in this function. The function caller SHOULD use function MsgReleaseFolderIdList to release pFolderIdList memory explicitlly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***rootFolderId*** indicates the root folder.

← ***folderType*** indicates the searched folder type.

→ ***pFolderIdList*** is a pointer to all the returned folder id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolderIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_TYPE Folder type is not valid for this operation.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.9 MSG_ERROR_T MsgGetFolderIdsByStorage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_STORAGE_ID_T *storageId*, MSG_FOLDER_ID_LIST_S * *pFolderIdList*)

Get all the folder id list on a specified storage.

This function gets all folder id list on a specified storage. The parameter `pFolderIdList` is defined outside this funtion, but the memory of its field `folderIds` is allocated in this function. The function caller should use function MsgReleaseFolderIdList to release pFolderIdList memory explicitlly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***storageId*** indicates the queried storage.

→ ***pFolderIdList*** is a pointer to the folder id list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolderIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_STORAGE Storage id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.10 MSG_ERROR_T MsgGetFolderParentId (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, MSG_FOLDER_ID_T * *pParentFolderId*)

Gets the parent folder id of a folder.

This function gets the parent folder id of a folder, and it will return 0 if it is a root folder.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** indicates the folder id which need to be check.

← ***pParentFolderId*** is a pointer to the returned parent folder id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pParentFolderId` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.11 MSG_ERROR_T MsgGetPredefinedFolders (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *rootFolderId*, MSG_FOLDER_S ** *ppInbox*, MSG_FOLDER_S ** *ppOutbox*, MSG_FOLDER_S ** *ppDraft*, MSG_FOLDER_S ** *ppSent*, MSG_FOLDER_S ** *ppDeleted*)

Gets the predefined folders for the specified root folder.

This function gets the predefined folders for the specified root folder. The predefined folders include inbox/outbox/sent/deleted/draft. If the value of output parameter is NULL, then function will not return that predefined folder information. For example, when calling this function `ppInbox` is NULL, then the function will not return Inbox predefined folder information. The memory of folder structure is allocated in this function. This function caller SHOULD use function MsgReleaseFolder to release the memory. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***rootFolderId*** indicates the searched root folder.

→ ***ppInbox*** returns the Inbox predefined folder information.

→ ***ppOutbox*** returns the Outbox predefined folder information.

→ ***ppDraft*** returns the Draft predefined folder information.

→ ***ppSent*** returns the Sent predefined folder information.

→ ***ppDeleted*** returns the Deleted predefined folder information.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.12 MSG_ERROR_T MsgMoveFolderToFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, MSG_FOLDER_ID_T *destFolderId*)

Moves a folder to the specified folder.

The function moves folder between folders.It will change the folder's parent folder to another folder. This function only works for user defined folder. The root folder and predefined folders cann't be changed.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** indicates the moved folder.

← ***destFolderId*** is the Id of the destination folder.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID `folderId` or `destFolderId` is invalid; `destFolderId` is the subfolder of `folderId`; the folder type of moved folder is predefined.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.13 MSG_ERROR_T MsgMoveFolderToStorage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, MSG_STORAGE_ID_T *destStorageId*)

Moves a folder to another storage.

The function moves a folder between storage. All the folder types, root folder, predefined folder, user defined folder, can be moved between different storage.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** indicates the moved folder.

← ***destStorageId*** is the Id of the destination storage.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID FOLDER id is invalid.

MSG_ERR_INVALID_STORAGE Storage id is invalid.

MSG_ERR_STORAGE_ERROR File system error.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_MAX_NUMBER_REACHED max number reached.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.12.2.14 MSG_ERROR_T MsgQueryFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, int *iBeginPos*, int *iCount*, MSG_FOLDER_ID_LIST_S * *pFolderIdList*)

Queries sub folder ids under the specified folder.

This function queries all the sub folder id list of the specified folder, which it only search the direct children of this folder and do not recursively search all the sub folders. The parameter `pFolderIdList` is defined outside this funtion, but the memory of its field `folderIds` is allocated in this function. The function caller SHOULD use function MsgReleaseFolderIdList to release `folderIds` memory explicitlly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** indicates the specific folder which is needed to query.

← ***iBeginPos*** is the index of the result set which satisfies the search condition. If iBeginPos is negative, the index begins from 0.

← ***iCount*** is the maximum number of folder information the caller wishes to receive. If iCount is negative, the method will get all satisfied folders.

→ ***pFolderIdList*** is a pointer to the returned folder id list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolderIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

For sample code, please refer to Folder Storage: Querying Folders

Since:

1.0

6.12.2.15 MSG_ERROR_T MsgQueryFolderRecursive (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *folderId*, int *iBeginPos*, int *iCount*, MSG_FOLDER_ID_LIST_S * *pFolderIdList*)

Recursively queries sub folder ids under the specified folder.

This function queries all the sub folder id list of the specified folder, which it recursively search all the sub folders. The parameter `pFolderIdList` is defined outside this funtion, but the memory of its field `folderIds` is allocated in this function. The function caller SHOULD use function MsgRelease-FolderIdList to release pFolderIdList memory explicitlly. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***folderId*** indicates the specific folder which is needed to query.

← ***iBeginPos*** is the index of the resultset which satisfies the search condition. If iBeginPos is negative, the index begins from 0.

← ***iCount*** is the maximum number of folder information the caller wishes to receive. If iCount is negative, this method will get all satisfied folders.

→ ***pFolderIdList*** is a pointer to the returned folder id list.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolderIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

For sample code, please refer to Folder Storage: Querying Folders

Since:

1.0

6.12.2.16 void MsgReleaseFolder (MSG_FOLDER_S * *pFolderInfo*)

Releases a memory block of a folder structure.

This function releases a memory of folder structure, which is allocated by function MsgGetFolder .

If pFolderInfo is NULL, nothing happens. If pFolderInfo is invalid, undefined behavior.

Parameters:

← ***pFolderInfo*** is a pointer to the folder memory block.

Since:

1.0

6.12.2.17 void MsgReleaseFolderIdList (MSG_FOLDER_ID_LIST_S * *pFolderIdList*)

Releases a memory block of folder id list.

This function frees the memory of pointer `folderIds` inside MSG_FOLDER_ID_LIST_S. If pFolderIdList or pFolderIdList->folderIds is NULL, nothing happens. If pFolderIdList or pFolderIdList->folderIds is invalid, undefined behavior.

Note:

This function does NOT free this structure itself.

Parameters:

← ***pFolderIdList*** is a pointer to the folder id list.

Since:

1.0

6.12.2.18 MSG_ERROR_T MsgUpdateFolder (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_FOLDER_S * *pFolder*)

Updates a folder.

This function updates a folder. The data fields, except for folderId, parentId, storageId, folderType, can be updated. Following fields of folder structure MUST be valid, otherwise the function will be failed:

`folderId`, `parentId`, `storageId`, `folderType`.

If the sortType is invalid, the default value MSG_SORT_BY_UNKNOWN will be set.

Parameters:

← ***hConnection*** is a connection handle.

← ***pFolder*** is a pointer to a structure which contains all updated information.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pFolder` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER The value of mandatory fields except folderId are invalid.

MSG_ERR_INVALID_FOLDER_ID This folder id doesn't exist in storage.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.13 Msg_Mapi_Storage_Manager.h File Reference

6.13.1 Detailed Description

Defines storage management functions in MAPI.

These functions can be divided into three categories:

The first is about connection operations.

The second is about storage registry operations.

The third is about query and sort operations.

Sample code:Storage General: Initializing Storage

```
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Storage_Registry.h"
#include "Msg_Mapi_QueryRule.h"
```

Typedefs

- typedef struct MessagingConnection * MSG_CONNECTION_HANDLE_T

  Defines a Connection handle.

Functions

- MSG_ERROR_T MsgCreateStoConnection (MSG_CONNECTION_HANDLE_T *hConnection)

  Creates a Connection handle.

- MSG_ERROR_T MsgDestroyStoConnection (MSG_CONNECTION_HANDLE_T hConnection)

  Destroys the Connection handle.

- MSG_ERROR_T MsgOpenStoConnection (MSG_CONNECTION_HANDLE_T hConnection)

  Opens a connection between application(client) and messaging framework(server).

- MSG_ERROR_T MsgCloseStoConnection (MSG_CONNECTION_HANDLE_T hConnection)

  Closes a connection between application(client) and messaging framework(server).

- MSG_ERROR_T MsgInitStorage ()

  Initializes the storages.

- MSG_ERROR_T MsgListStorageRegistry (MSG_CONNECTION_HANDLE_T hConnection, MSG_STORAGE_ID_LIST_S *pStorageIdList)

  Returns all registered storage ids.

- void MsgReleaseStorageIdList (MSG_STORAGE_ID_LIST_S *pStorageIdList)

  Releases the memory of storage Id List.

- MSG_ERROR_T MsgGetStorageRegistry (MSG_CONNECTION_HANDLE_T hConnection, MSG_STORAGE_ID_T storageId, MSG_STORAGE_REGISTRY_S **ppStorageRegInfo)

Gets a storage from registry by storage id.

- MSG_ERROR_T MsgGetMediaTypeByStorageId (MSG_CONNECTION_HANDLE_T hConnection, MSG_STORAGE_ID_T storageId, MSG_MEDIA_TYPE_T *pMediaType)

  Gets a media type from registry by storage id.

- void MsgReleaseStorageRegistry (MSG_STORAGE_REGISTRY_S *pStorageRegInfo)

  Releases the memory of MSG_STORAGE_REGISTER_INFO_S structure.

- MSG_ERROR_T MsgListSortType (MSG_CONNECTION_HANDLE_T hConnection, MSG_SORT_TYPE_LIST_S *pSortTypeList)

  Gets a sort type list from configuration files.

- void MsgReleaseSortTypeList (MSG_SORT_TYPE_LIST_S *pSortTypeList)

  Releases the memory of sort type list.

- MSG_ERROR_T MsgListQueryId (MSG_CONNECTION_HANDLE_T hConnection, MSG_QUERY_ID_LIST_S *pQueryIdList)

  Gets a query Id list from configuration files.

- void MsgReleaseQueryIdList (MSG_QUERY_ID_LIST_S *pQueryIdList)

  Releases the memory of query Id list.

- MSG_ERROR_T MsgGetFwStorageStatus (MSG_STORAGE_ID_T storageId, unsigned long long *totalSize, unsigned long long *usedSize, MSG_SIZE_UNIT_T *unit)

  Gets framework storage status by storage id.

- MSG_ERROR_T MsgGetMsgStorageStatus (MSG_STORAGE_ID_T storageId, const MSG_MESSAGE_TYPE_S *msgType, unsigned long long *totalSize, unsigned long long *usedSize, MSG_SIZE_UNIT_T *unit)

  Gets specific message storage status by storage id and message type.

- MSG_PLUGIN_ERROR_T MsgGetStoragePluginError (MSG_CONNECTION_HANDLE_T hConnection)

  Get storage specific plugin error.

6.13.2 Typedef Documentation

6.13.2.1 typedef struct MessagingConnection* MSG_CONNECTION_HANDLE_T

Defines a Connection handle.

For details of Connection, please refer to Connection

6.13.3 Function Documentation

6.13.3.1 MSG_ERROR_T MsgCloseStoConnection (MSG_CONNECTION_HANDLE_T *hConnection*)

Closes a connection between application(client) and messaging framework(server).

When this function returns successfully, the communication between the application and messaging framework will be unavailable.

If hConnection is NULL, this function does nothing and returns MSG_ERR_INVALID_CONNECTION.

For details of Connection, please refer to Connection

Parameters:

← ***hConnection*** is a Connection handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

Please refer to Storage General: Initializing Storage

Since:

1.0

6.13.3.2 MSG_ERROR_T MsgCreateStoConnection (MSG_CONNECTION_HANDLE_T * *hConnection*)

Creates a Connection handle.

This is the only way for applications to get a Connection handle. Such operations as adding, deleting and updating must be bound with a connection handle.

For details of Connection, please refer to Connection

Parameters:

→ ***hConnection*** is a pointer to a Connection handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER Pointer is NULL.

MSG_ERR_MEMORY_ERROR memory is error.

Please refer to Storage General: Initializing Storage.

Since:

1.0

6.13.3.3 MSG_ERROR_T MsgDestroyStoConnection (MSG_CONNECTION_HANDLE_T *hConnection*)

Destroys the Connection handle.

After the Connection is destroyed, it can not be used any more. For each created Connection, please make sure that it should be destroyed when it's not used any more. If the input parameter is NULL, this function does nothing and returns MSG_ERR_INVALID_CONNECTION.

For details of Connection, please refer to Connection

Parameters:

← ***hConnection*** is a Connection handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

Since:

1.0

6.13.3.4 MSG_ERROR_T MsgGetFwStorageStatus (MSG_STORAGE_ID_T *storageId*, unsigned long long * *totalSize*, unsigned long long * *usedSize*, MSG_SIZE_UNIT_T * *unit*)

Gets framework storage status by storage id.

This function is to get total size and used size of the specified framework storage.

Parameters:

← ***storageId*** is an integer value indicating which storage.

→ ***totalSize*** the total size in this storage.

→ ***usedSize*** used size in this storage.

→ ***unit*** the unit of the size.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `totalSize` is NULL or `usedSize` is NULL or unit is NULL.

MSG_ERR_INVALID_STORAGE Storage id is invalid

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.13.3.5 MSG_ERROR_T MsgGetMediaTypeByStorageId (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_STORAGE_ID_T *storageId*, MSG_MEDIA_TYPE_T * *pMediaType*)

Gets a media type from registry by storage id.

Parameters:

← ***hConnection*** is a Connection handle.

← ***storageId*** is an integer value indicating which storage.

→ ***pMediaType*** is an enumeration value indicating which media .

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMediaType` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_STORAGE Storage id is invalid

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.13.3.6 MSG_ERROR_T MsgGetMsgStorageStatus (MSG_STORAGE_ID_T *storageId*, const MSG_MESSAGE_TYPE_S * *msgType*, unsigned long long * *totalSize*, unsigned long long * *usedSize*, MSG_SIZE_UNIT_T * *unit*)

Gets specific message storage status by storage id and message type.

This function is to get total size and used size of the specified message storage.

Parameters:

← ***storageId*** is an integer value indicating which storage.

← ***msgType*** is the specific message type

→ ***totalSize*** the total size in this storage.

→ ***usedSize*** used size in this storage.

→ ***unit*** the unit of the size.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `msgType` is NULL or `totalSize` is NULL or `usedSize` is NULL or `unit` is NULL.

MSG_ERR_INVALID_STORAGE Storage id is invalid

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.13.3.7 MSG_PLUGIN_ERROR_T MsgGetStoragePluginError (MSG_CONNECTION_-HANDLE_T *hConnection*)

Get storage specific plugin error.

This function is to get the specific plugin error. In case that some functions return MSG_ERR_PLUGIN, applications can call this function immediately to get specific error code of the plugins. If hConnection is invalid, this function will return 0.

Parameters:

← ***hConnection*** is a pointer to an established connection handle.

Returns:

MSG_PLUGIN_ERROR_T

6.13.3.8 MSG_ERROR_T MsgGetStorageRegistry (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_STORAGE_ID_T *storageId*, MSG_STORAGE_REGISTRY_S ** *ppStorageRegInfo*)

Gets a storage from registry by storage id.

This function allocates the memory of MSG_STORAGE_REGISTRY_S structure. Applications need to call MsgReleaseStorageRegistry to release the memory after using the structure.

Parameters:

← ***hConnection*** is a Connection handle.

← ***storageId*** is an integer value indicating which storage.

→ ***ppStorageRegInfo*** is a pointer to a struct pointer .

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `ppStorageRegInfo` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_STORAGE Storage id is invalid

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.13.3.9 MSG_ERROR_T MsgInitStorage ()

Initializes the storages.

If there is data in storage, all the data will be cleaned up and reset with default values, If there is no data in storage, all the data will be created and set with default values.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_STORAGE_ERROR Storage error.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.13.3.10 MSG_ERROR_T MsgListQueryId (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_QUERY_ID_LIST_S * *pQueryIdList*)

Gets a query Id list from configuration files.

Applications need to call MsgReleaseQueryIdList to release the memory of queryIds in the structure.

Parameters:

← ***hConnection*** is a Connection handle.

→ ***pQueryIdList*** is a pointer to a MSG_QUERY_ID_LIST_S struct .

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.
MSG_ERR_NULL_POINTER `pQueryIdList` is NULL.
MSG_ERR_INVALID_CONNECTION Connection is invalid.
MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.
MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.
MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.13.3.11 MSG_ERROR_T MsgListSortType (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_SORT_TYPE_LIST_S * *pSortTypeList*)

Gets a sort type list from configuration files.

Applications need to call MsgReleaseSortTypeList to release the memory of sortTypes in the structure.

Parameters:

← ***hConnection*** is a Connection handle.
→ ***pSortTypeList*** is a pointer to a MSG_SORT_TYPE_LIST_S structure .

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.
MSG_ERR_NULL_POINTER `pSortTypeList` is NULL.
MSG_ERR_INVALID_CONNECTION Connection is invalid.
MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.
MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.
MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.13.3.12 MSG_ERROR_T MsgListStorageRegistry (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_STORAGE_ID_LIST_S * *pStorageIdList*)

Returns all registered storage ids.

Applications need to allocate the structure `pStorageIdList`.

When this function returns successfully, the registered storage ids will be filled in pStorageIdList->pStorageIds. Applications need to call MsgReleaseStorageIdList to release the memory of pStorageIdList->pStorageIds.

Parameters:

← ***hConnection*** is a Connection handle.

→ ***pStorageIdList*** is a pointer to an MSG_STORAGE_ID_LIST_S structure.Indicate which storages will be supported.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pStorageIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

Since:

1.0

6.13.3.13 MSG_ERROR_T MsgOpenStoConnection (MSG_CONNECTION_HANDLE_T *hConnection*)

Opens a connection between application(client) and messaging framework(server).

When this function returns successfully, applications can do the operations such as adding, deleting and updating etc.

For details of Connection, please refer to Connection

Parameters:

← ***hConnection*** is a pointer to a Connection handle.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

Please refer to Storage General: Initializing Storage

Since:

1.0

6.13.3.14 void MsgReleaseQueryIdList (MSG_QUERY_ID_LIST_S * *pQueryIdList*)

Releases the memory of query Id list.

This function releases the memory of queryIds in MSG_QUERY_ID_LIST_S structure. If pQueryIdList or pQueryIdList->queryIds is NULL, nothing happens. If pQueryIdList or pQueryIdList->queryIds is invalid, undefined behavior.

Note:

This function only frees the list member inside the structure. It doesn't free the structure itself.

Parameters:

← ***pQueryIdList*** is a pointer to an MSG_QUERY_ID_LIST_S structure.

Since:

1.0

6.13.3.15 void MsgReleaseSortTypeList (MSG_SORT_TYPE_LIST_S * *pSortTypeList*)

Releases the memory of sort type list.

This function releases the memory of sortTypes in MSG_SORT_TYPE_LIST_S structure.

If pSortTypeList or pSortTypeList->sortTypes is NULL, nothing happens. If pSortTypeList or pSortTypeList->sortTypes is invalid, undefined behavior.

Note:

This function only frees the list member inside the structure. It doesn't free the structure itself.

Parameters:

← ***pSortTypeList*** is a pointer to an MSG_SORT_TYPE_LIST_S structure.

Since:

1.0

6.13.3.16 void MsgReleaseStorageIdList (MSG_STORAGE_ID_LIST_S * *pStorageIdList*)

Releases the memory of storage Id List.

This function releases the memory of pStorageIds in MSG_STORAGE_ID_LIST_S structure.

If pStorageIdList or pStorageIdList->pStorageIds is NULL, nothing happens. If pStorageIdList or pStorageIdList->pStorageIds is invalid, undefined behavior.

Note:

This function only frees the list pointer member inside the structure. It doesn't free the structure itself.

Parameters:

← ***pStorageIdList*** is a pointer to an MSG_STORAGE_ID_LIST_S structure.Indicate which storages will be released.

Since:

1.0

6.13.3.17 void MsgReleaseStorageRegistry (MSG_STORAGE_REGISTRY_S * *pStorageRegInfo*)

Releases the memory of MSG_STORAGE_REGISTER_INFO_S structure.

If pStorageRegInfo is NULL, nothing happens. If pStorageRegInfo is invalid, undefined behavior.

Parameters:

← ***pStorageRegInfo*** is a pointer to an MSG_STORAGE_REGISTRY_S structure.

Since:

1.0

6.14 Msg_Mapi_Storage_Message.h File Reference

6.14.1 Detailed Description

Defines message storage related functions in MAPI.

The file includes all message storage functions, such as adding, updating, deleting etc.

```
#include "Msg_Mapi_Types.h"
#include "Msg_Mapi_Message.h"
#include "Msg_Mapi_Storage_Manager.h"
#include "Msg_Mapi_QueryRule.h"
```

Functions

- MSG_ERROR_T MsgAddMessage (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_S *pMsg)

  Adds a message into storage.

- MSG_ERROR_T MsgUpdateMessage (MSG_CONNECTION_HANDLE_T hConnection, const MSG_MESSAGE_S *pMsg)

  Updates a message in storage.

- MSG_ERROR_T MsgDeleteMessage (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_ID_T MsgId)

  Deletes a message by message idn.

- MSG_ERROR_T MsgDeleteAllMessageInFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId)

  Deletes all messages in the specified folder.

- MSG_ERROR_T MsgMoveMessageToFolder (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_ID_T MsgId, MSG_FOLDER_ID_T DesFolderId)

  Moves a message to the specified folder.

- MSG_ERROR_T MsgMoveMessageToStorage (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_ID_T MsgId, MSG_STORAGE_ID_T DestStorageId)

  Moves a message to another storage.

- MSG_ERROR_T MsgCountMessage (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId, const MSG_MESSAGE_TYPE_S *pMsgType, int *pCount)

  Gets the number of messages under the specified folder by message type.

- MSG_ERROR_T MsgCountMessageRecursive (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId, const MSG_MESSAGE_TYPE_S *pMsgType, int *pCount)

  Gets the number of messages, both directly and indirectly, under the specified folder by message type.

- MSG_ERROR_T MsgCountUnreadMessage (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId, const MSG_MESSAGE_TYPE_S *pMsgType, int *pCount)

  Gets the number of unread messages under the specified folder by message type.

- MSG_ERROR_T MsgCountUnreadMessageRecursive (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId, const MSG_MESSAGE_TYPE_S *pMsgType, int *pCount)

  Gets the number of unread messages, both directly and indirectly, under the specified folder by message type.

- MSG_ERROR_T MsgQueryMessage (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId, int iBeginIndex, int iCount, const MSG_SORT_RULE_S *pSortRule, const MSG_QUERY_LIST_S *pQueryList, MSG_MESSAGE_ID_LIST_S *pMsgIdList)

  Gets the matched message ids under the specified folder.

- MSG_ERROR_T MsgQueryMessageRecursive (MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T FolderId, int iBeginIndex, int iCount, const MSG_SORT_RULE_S *pSortRule, const MSG_QUERY_LIST_S *pQueryList, MSG_MESSAGE_ID_LIST_S *pMsgIdList)

  Gets the matched message ids, both directly and indirectly, under the specified folder.

- MSG_ERROR_T MsgGetMessage (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_ID_T MsgId, MSG_MESSAGE_S **ppMsg)

  Returns the message detail information.

- MSG_ERROR_T MsgGetMessageCommInfo (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_ID_T MsgId, MSG_MESSAGE_S **ppMsg)

  Returns the message common information.

- MSG_ERROR_T MsgGetMessageParentId (MSG_CONNECTION_HANDLE_T hConnection, MSG_MESSAGE_ID_T MsgId, MSG_FOLDER_ID_T *pParentFolderId)

  Returns the folder id in which the message located.

- void MsgReleaseMessage (MSG_MESSAGE_S *pMsg)

  Frees the memory of MSG_MESSAGE_S allocated in MsgGetMessage If pMsg is NULL, nothing happens. If pMsg is invalid, undefined behavior.

- void MsgReleaseMessageCommInfo (MSG_MESSAGE_S *pMsg)

  Frees the memory of MSG_MESSAGE_S allocated in MsgGetMessageCommInfo If pMsg is NULL, nothing happens. If pMsg is invalid, undefined behavior.

- void MsgReleaseMessageIdList (MSG_MESSAGE_ID_LIST_S *pMsgIdList)

  Frees the memory of pointer inside MSG_MESSAGE_ID_LIST_S allocated in MsgQueryMessage, MsgQueryMessageRecursive and MsgGetMessageIdsByStorage.

- MSG_ERROR_T MsgGetMessageIdsByStorage (MSG_CONNECTION_HANDLE_T hConnection, MSG_STORAGE_ID_T StorageId, MSG_MESSAGE_ID_LIST_S *pMsgIds)

  Returns the message Ids by storage Id.

6.14.2 Function Documentation

6.14.2.1 MSG_ERROR_T MsgAddMessage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_S * *pMsg*)

Adds a message into storage.

Mandatory fields of message structure MUST be valid, otherwise the function will be failed. The mandatory fields for adding operation are `parentId`, `msgType`, `storageId`.

If `storageId` is MSG_STORAGE_DEFAULT, the message will be added to the `defaultMsgStorageId` in its associated account.

Parameters:

← ***hConnection*** is a connection handle.

↔ ***pMsg*** is a pointer to a MSG_MESSAGE_S structure. pMsg->msgId will be set to a valid value in storage when this function is returned successfully.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsg` is NULL .

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE Mandatory fields of message are invalid.

MSG_ERR_STORAGE_FULL Storage is FULL.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_MAX_NUMBER_REACHED max number reached

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Please refer to Message Storage: Creating and Storing a Message.

Since:

1.0

6.14.2.2 MSG_ERROR_T MsgCountMessage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*, const MSG_MESSAGE_TYPE_S * *pMsgType*, int * *pCount*)

Gets the number of messages under the specified folder by message type.

Note:

If `pMsgType` is NULL, it will return the messages number of all the message types.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is a Id of folder to be counted.

← ***pMsgType*** is a pointer to MSG_MESSAGE_TYPE_S structure.

→ ***pCount*** is a integer pointer.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL .

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE Message type is invalid.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.3 MSG_ERROR_T MsgCountMessageRecursive (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*, const MSG_MESSAGE_TYPE_S * *pMsgType*, int * *pCount*)

Gets the number of messages, both directly and indirectly, under the specified folder by message type.

Note:

If `pMsgType` is NULL, it will return the messages number of all the message types.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is querying folder Id.

← ***pMsgType*** is a pointer to MSG_MESSAGE_TYPE_S structure.

→ ***pCount*** is an integer pointer.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL .

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE Message type is invalid.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.4 MSG_ERROR_T MsgCountUnreadMessage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*, const MSG_MESSAGE_TYPE_S * *pMsgType*, int * *pCount*)

Gets the number of unread messages under the specified folder by message type.

Note:

If `pMsgType` is NULL, it will return the messages number of all the message types.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is querying folder Id.

← ***pMsgType*** is a pointer to MSG_MESSAGE_TYPE_S structure.

→ ***pCount*** is an integer pointer.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL .

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE Message type is invalid.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.5 MSG_ERROR_T MsgCountUnreadMessageRecursive (MSG_CONNECTION_-HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*, const MSG_MESSAGE_TYPE_S * *pMsgType*, int * *pCount*)

Gets the number of unread messages, both directly and indirectly, under the specified folder by message type.

Note:

If `pMsgType` is NULL, it will return the messages number of all the message types.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is a querying folder Id.

← ***pMsgType*** is a pointer to MSG_MESSAGE_TYPE_S structure.

→ ***pCount*** is an integer pointer.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pCount` is NULL .

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MSG_TYPE Message type is invalid.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.6 MSG_ERROR_T MsgDeleteAllMessageInFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*)

Deletes all messages in the specified folder.

This function is to delete all messages in the specified folder,but it is not to delete the specified folder's subfolders and the messages in the subfolders.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is a folder Id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.7 MSG_ERROR_T MsgDeleteMessage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_ID_T *MsgId*)

Deletes a message by message idn.

Parameters:

← ***hConnection*** is a connection handle.

← ***MsgId*** indicates to be deleted message's Id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE_ID Message id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.8 MSG_ERROR_T MsgGetMessage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_ID_T *MsgId*, MSG_MESSAGE_S ** *ppMsg*)

Returns the message detail information.

This function returns the message detail information including both common extension data (`pCommExtData`) and specific message extension data (`pExtData`).

Note:

Message memory will be allocated inside this function. Applications need to call MsgReleaseMessage to free the memory. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***MsgId*** indicates to be returned message's Id.

→ ***ppMsg*** is a pointer to a message pointer.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `ppMsg` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE_ID Message id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Please refer to Message Storage: Retrieving Message List and Entity.

Since:

1.0

6.14.2.9 MSG_ERROR_T MsgGetMessageCommInfo (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_ID_T *MsgId*, MSG_MESSAGE_S ** *ppMsg*)

Returns the message common information.

This function returns the message common information including common extension data (pCommExtData) without specific message extension data (pExtData).

Note:

Message memory will be allocated inside this function. Applications need to call MsgReleaseMessageCommInfo to free the memory. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***MsgId*** indicates to be returned message's Id.

→ ***ppMsg*** is a pointer to a message pointer.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `ppMsg` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE_ID Message id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.10 MSG_ERROR_T MsgGetMessageIdsByStorage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_STORAGE_ID_T *StorageId*, MSG_MESSAGE_ID_LIST_S * *pMsgIds*)

Returns the message Ids by storage Id.

Applications need to allocate the structure `pMsgIds`.

When this function returns successfully, the storage ids will be filled in pMsgIds->msgIds. Applications need to call MsgReleaseMessageIdList to release the memory of pMsgIds->msgIds. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***StorageId*** is querying storage Id.

→ ***pMsgIds*** is a pointer to a MSG_MESSAGE_ID_LIST_S structure.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsgIds` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_STORAGE Message id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.11 MSG_ERROR_T MsgGetMessageParentId (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_ID_T *MsgId*, MSG_FOLDER_ID_T * *pParentFolderId*)

Returns the folder id in which the message located.

Parameters:

← ***hConnection*** is a connection handle.

← ***MsgId*** indicates which message should be gotten.

→ ***pParentFolderId*** a pointer to the parent folder id.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pParentFolderId` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID Message id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.12 MSG_ERROR_T MsgMoveMessageToFolder (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_ID_T *MsgId*, MSG_FOLDER_ID_T *DesFolderId*)

Moves a message to the specified folder.

The function moves a message between folders.

Parameters:

← ***hConnection*** is a connection handle.

← ***MsgId*** is the Id of the message to be moved.

← ***DesFolderId*** is the Id of the destination folder.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE_ID Message id is invalid.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.13 MSG_ERROR_T MsgMoveMessageToStorage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_MESSAGE_ID_T *MsgId*, MSG_STORAGE_ID_T *DestStorageId*)

Moves a message to another storage.

The function moves a message between storages.

Parameters:

← ***hConnection*** is a connection handle.

← ***MsgId*** is the Id of the message to be moved.

← ***DestStorageId*** is the Id of the destination storage.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE_ID Message id is invalid.

MSG_ERR_INVALID_STORAGE Storage id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_MAX_NUMBER_REACHED max number reached

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.14.2.14 MSG_ERROR_T MsgQueryMessage (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*, int *iBeginIndex*, int *iCount*, const MSG_SORT_RULE_S * *pSortRule*, const MSG_QUERY_LIST_S * *pQueryList*, MSG_MESSAGE_ID_LIST_S * *pMsgIdList*)

Gets the matched message ids under the specified folder.

If `pQueryList` is NULL,the function will return all message in this folder. If `pSortRule` is NULL, the sort type is depended on underlying storage.

Note:

The memory at MSG_MESSAGE_ID_S of MSG_MESSAGE_ID_LIST_S will be allocated inside this method, which need be released by calling MsgReleaseMessageIdList. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is querying folder Id.

← ***iBeginIndex*** is an integer value. Indicate start position. If iBeginIndex is negative, index begins from 0.

← ***iCount*** is the maximum number of message information the caller wishes to receive. If iCount is negative, this method will get all satisfied messages.

← ***pSortRule*** is a pointer to MSG_SORT_RULE_S structure. Indicate to use which sort rule.

← ***pQueryList*** is a pointer MSG_QUERY_LIST_S structure. Indicate QueryRule,for query rule description,refer to MSG_QUERY_LIST_S.

→ ***pMsgIdList*** is a pointer to a MSG_MESSAGE_ID_LIST_S structure.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsgIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_INVALID_SORT_TYPE Sort type is invalid.

MSG_ERR_INVALID_QUERY_ID Query id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Please refer to Message Storage: Retrieving Message List and Entity.

Since:

1.0

6.14.2.15 MSG_ERROR_T MsgQueryMessageRecursive (MSG_CONNECTION_HANDLE_T *hConnection*, MSG_FOLDER_ID_T *FolderId*, int *iBeginIndex*, int *iCount*, const MSG_SORT_RULE_S * *pSortRule*, const MSG_QUERY_LIST_S * *pQueryList*, MSG_MESSAGE_ID_LIST_S * *pMsgIdList*)

Gets the matched message ids, both directly and indirectly, under the specified folder.

If `pQueryList` is NULL,the function will return all message in this folder. If `pSortRule` is NULL, the sort type is depended on underlying storage.

Note:

The memory at MSG_MESSAGE_ID_S of MSG_MESSAGE_ID_LIST_S will be allocated inside this method, which need be released by calling MsgReleaseMessageIdList. However if this function is failed, the memory is NOT allocated in this function.

Parameters:

← ***hConnection*** is a connection handle.

← ***FolderId*** is querying folder Id.

← ***iBeginIndex*** is an integer value. Indicate start position. If iBeginIndex is negative, index begins from 0.

← ***iCount*** is the maximum number of message information the caller wishes to receive. If iCount is negative, this method will get all satisfied messages.

← ***pSortRule*** is a pointer to MSG_SORT_RULE_S structure. Indicate to use which sort rule.

← ***pQueryList*** is a pointer to MSG_QUERY_LIST_S structure. Indicate QueryRule,for query rule description,refer to MSG_QUERY_LIST_S.

→ ***pMsgIdList*** is a pointer to a MSG_MESSAGE_ID_LIST_S structure.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsgIdList` is NULL.

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_FOLDER_ID Folder id is invalid.

MSG_ERR_INVALID_SORT_TYPE Sort type is invalid.

MSG_ERR_INVALID_QUERY_ID Query id is invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Please refer to Message Storage: Retrieving Message List and Entity.

Since:

1.0

6.14.2.16 void MsgReleaseMessage (MSG_MESSAGE_S * *pMsg*)

Frees the memory of MSG_MESSAGE_S allocated in MsgGetMessage If pMsg is NULL, nothing happens. If pMsg is invalid, undefined behavior.

Parameters:

← ***pMsg*** is a pointer to an MSG_MESSAGE_S structure.

Since:

1.0

6.14.2.17 void MsgReleaseMessageCommInfo (MSG_MESSAGE_S * *pMsg*)

Frees the memory of MSG_MESSAGE_S allocated in MsgGetMessageCommInfo If pMsg is NULL, nothing happens. If pMsg is invalid, undefined behavior.

Parameters:

← ***pMsg*** is a pointer to an MSG_MESSAGE_S structure.

Since:

1.0

6.14.2.18 void MsgReleaseMessageIdList (MSG_MESSAGE_ID_LIST_S * *pMsgIdList*)

Frees the memory of pointer inside MSG_MESSAGE_ID_LIST_S allocated in MsgQueryMessage, MsgQueryMessageRecursive and MsgGetMessageIdsByStorage.

This function releases the memory of msgIds in MSG_MESSAGE_ID_LIST_S structure. If pMsgIdList or pMsgIdList->msgIds is NULL, nothing happens. If pMsgIdList or pMsgIdList->msgIds is invalid, undefined behavior.

Note:

This function does not free this sturcture itself.

Parameters:

← ***pMsgIdList*** is a pointer to an MSG_MESSAGE_ID_LIST_S structure.

Since:

1.0

6.14.2.19 MSG_ERROR_T MsgUpdateMessage (MSG_CONNECTION_HANDLE_T *hConnection*, const MSG_MESSAGE_S * *pMsg*)

Updates a message in storage.

The function is to update message data for the message indentified by the given msgId as long as the given values are valid.

Following fields of message structure MUST be valid, otherwise the function will be failed. `msgId`, `parentId`, `msgType`, `storageId`.

Note:

pMsg->msgId MUST NOT be updated,because pMsg->msgId is a unique Id on platform.
pMsg->parentId MUST NOT be updated. The function will return MSG_ERR_INVALID_-MESSAGE, if inputting a new pMsg->parentId. If applications want to move message between folders, applications SHOULD call MsgMoveMessageToFolder.
pMsg->storageId MUST NOT be updated. The function will return MSG_ERR_INVALID_-MESSAGE,if inputting a new pMsg->storageId. If applications want to move message between storages, applications SHOULD call MsgMoveMessageToStorage.

Parameters:

← ***hConnection*** is a connection handle.

← ***pMsg*** is a const pointer to a MSG_MESSAGE_S structure.

Returns:

MSG_ERROR_T

Return values:

MSG_SUCCESS Succeeds in operation.

MSG_ERR_NULL_POINTER `pMsg` is NULL .

MSG_ERR_INVALID_CONNECTION Connection is invalid.

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected.

MSG_ERR_INVALID_MESSAGE Mandatory fields of message are invalid.

MSG_ERR_COMMUNICATION_ERROR communication between client and server is error.

MSG_ERR_MEMORY_ERROR memory is error.

MSG_ERR_PLUGIN the generic error code for plugin. Call MsgGetStoragePluginError() to get detail error code.

Since:

1.0

6.15 Msg_Mapi_Storage_Registry.h File Reference

6.15.1 Detailed Description

Defines the storage registry structure.

```
#include "Msg_Mapi_Types.h"
#include <stdlib.h>
```

Data Structures

- struct _MSG_STORAGE_REGISTRY_S

  Represents a storage registry entry.

Typedefs

- typedef struct _MSG_STORAGE_REGISTRY_S MSG_STORAGE_REGISTRY_S

6.15.2 Typedef Documentation

6.15.2.1 typedef struct _MSG_STORAGE_REGISTRY_S MSG_STORAGE_REGISTRY_S

6.16 Msg_Mapi_Types.h File Reference

6.16.1 Detailed Description

Defines general types.

Data Structures

- struct _MSG_MESSAGE_TYPE_S
  Represents a message type.
- struct _MSG_MESSAGE_TYPE_LIST_S
  Represents a message types list.
- struct _MSG_STORAGE_ID_LIST_S
  Represents a storage id list.
- struct _MSG_STRING_S
  Represents a string.
- struct _MSG_MESSAGE_ID_LIST_S
  Represents a message id list.
- struct _MSG_FOLDER_ID_LIST_S
  Represents a folder id list.
- struct _MSG_ACCOUNT_ID_LIST_S
  Represents an account id list.

Defines

- #define bool unsigned char
  Defines bool type.
- #define true 1
  Defines true value.
- #define false 0
  Defines false value.
- #define MSG_INVALID_MESSAGE_ID 0xffffffff
  Defines invalid id for message.
- #define MSG_INVALID_ACCOUNT_ID 0xffff
  Defines invalid id for account.
- #define MSG_INVALID_FOLDER_ID 0xffff
  Defines invalid id for folder.

- #define MSG_DEFAULT_ACCOUNT_ID 0

  Defines default account id.

- #define MSG_DEFAULT_FOLDER_ID 0

  Defines default folder id.

- #define MSG_ERROR_BEGIN (-255)

  Defines begin value for error.

Typedefs

- typedef int MSG_MAIN_TYPE_T

  Represents a message main type.

- typedef int MSG_SUB_TYPE_T

  Represents a message sub type.

- typedef struct _MSG_MESSAGE_TYPE_S MSG_MESSAGE_TYPE_S
- typedef struct _MSG_MESSAGE_TYPE_LIST_S MSG_MESSAGE_TYPE_LIST_S
- typedef int MSG_ERROR_T

  Represents an error code.

- typedef unsigned int MSG_PLUGIN_ERROR_T

  Error code specific for plugins.

- typedef int MSG_SESSION_PRIORITY_T

  Represents a session priority.

- typedef int MSG_MESSAGE_PRIORITY_T

  Represents a message priority.

- typedef int MSG_STATUS_T

  Represents a status.

- typedef int MSG_REQUEST_TYPE_T

  Represents a request type.

- typedef unsigned int MSG_REQUEST_ID_T

  Represents a request id, which is unique for each request.

- typedef unsigned int MSG_STORAGE_ID_T

  Represents a Storage id.

- typedef unsigned int MSG_MEDIA_TYPE_T

  Represents a media type.

- typedef unsigned int MSG_FOLDER_TYPE_T

Represents a folder type.

- typedef struct _MSG_STORAGE_ID_LIST_S MSG_STORAGE_ID_LIST_S
- typedef int MSG_CONTACT_TYPE_T

  Represents a contact type.

- typedef int MSG_ENCODE_TYPE_T

  Represents an encoding type.

- typedef struct _MSG_STRING_S MSG_STRING_S
- typedef unsigned long MSG_MESSAGE_ID_T

  Represents a messsage id.

- typedef unsigned int MSG_FOLDER_ID_T

  Represents a folder id.

- typedef unsigned int MSG_ACCOUNT_ID_T

  Represents an account id.

- typedef struct _MSG_MESSAGE_ID_LIST_S MSG_MESSAGE_ID_LIST_S
- typedef struct _MSG_FOLDER_ID_LIST_S MSG_FOLDER_ID_LIST_S
- typedef struct _MSG_ACCOUNT_ID_LIST_S MSG_ACCOUNT_ID_LIST_S
- typedef int MSG_SIZE_UNIT_T

  Represents the unit of the size.

Enumerations

- enum _MSG_MAIN_TYPE_E {

  MSG_MESSAGE_TYPE_UNKNOWN, MSG_MESSAGE_TYPE_MMS, MSG_MESSAGE_TYPE_EMAIL, MSG_MESSAGE_TYPE_SMS,

  MSG_MESSAGE_TYPE_IMODE }

  Represents the values of message main types.

- enum _MSG_ERROR_E {

  MSG_SUCCESS = 0, MSG_ERR_FS_ERROR = MSG_ERROR_BEGIN - 1, MSG_ERR_FS_FULL = MSG_ERROR_BEGIN - 2, MSG_ERR_INVALID_STORAGE = MSG_ERROR_BEGIN - 3,

  MSG_ERR_INVALID_MSG_TYPE = MSG_ERROR_BEGIN - 4, MSG_ERR_INVALID_SORT_TYPE = MSG_ERROR_BEGIN - 5, MSG_ERR_INVALID_FOLDER_ID = MSG_ERROR_BEGIN - 6, MSG_ERR_INVALID_STORAGE_REG = MSG_ERROR_BEGIN - 7,

  MSG_ERR_INVALID_CONNECTION = MSG_ERROR_BEGIN - 8, MSG_ERR_NULL_POINTER = MSG_ERROR_BEGIN - 9, MSG_ERR_INVALID_MESSAGE_ID = MSG_ERROR_BEGIN - 10, MSG_ERR_INVALID_ACCOUNT_ID = MSG_ERROR_BEGIN - 11,

  MSG_ERR_INVALID_QUERY_ID = MSG_ERROR_BEGIN - 12, MSG_ERR_SESSION_NOT_CONNECTED = MSG_ERROR_BEGIN - 13, MSG_ERR_INVALID_SESSION_HANDLE = MSG_ERROR_BEGIN - 14, MSG_ERR_PERMISSION_DENIED = MSG_ERROR_BEGIN - 15,

MSG_ERR_STORAGE_NOT_INIT = MSG_ERROR_BEGIN - 16, MSG_ERR_STORAGE_REG_NOT_INIT = MSG_ERROR_BEGIN - 17, MSG_ERR_INVALID_SESSION_PRIORITY = MSG_ERROR_BEGIN - 18, MSG_ERR_INVALID_FILTER_ID = MSG_ERROR_BEGIN - 19,

MSG_ERR_INVALID_REQUEST = MSG_ERROR_BEGIN - 20, MSG_ERR_INVALID_FOLDER_TYPE = MSG_ERROR_BEGIN - 21, MSG_ERR_INVALID_FOLDER = MSG_ERROR_BEGIN - 22, MSG_ERR_NOT_EMPTY_FOLDER = MSG_ERROR_BEGIN - 23,

MSG_ERR_INVALID_ACCOUNT = MSG_ERROR_BEGIN - 24, MSG_ERR_CONNECTION_NOT_CONNECT = MSG_ERROR_BEGIN - 25, MSG_ERR_INVALID_MESSAGE = MSG_ERROR_BEGIN - 26, MSG_ERR_STORAGE_FULL = MSG_ERROR_BEGIN - 27,

MSG_ERR_ROOTFOLDER_INUSE = MSG_ERROR_BEGIN - 28, MSG_ERR_STORAGE_ERROR = MSG_ERROR_BEGIN - 29, MSG_ERR_MEMORY_ERROR = MSG_ERROR_BEGIN - 30, MSG_ERR_COMMUNICATION_ERROR = MSG_ERROR_BEGIN - 31,

MSG_ERR_MAX_NUMBER_REACHED = MSG_ERROR_BEGIN - 32, MSG_ERR_INVALID_REQUEST_ID = MSG_ERROR_BEGIN - 33, MSG_ERR_UNKNOWN = MSG_ERROR_BEGIN - 0x00008000, MSG_ERR_FRAMEWORK_RESERVE_BEGIN = MSG_ERROR_BEGIN - 0x00010000,

MSG_ERR_FRAMEWORK_RESERVE_END = MSG_ERROR_BEGIN - 0x00020000, MSG_ERR_PLUGIN = MSG_ERROR_BEGIN - 0x08000000, MSG_ERR_PLUGIN_DEF_BEGIN_VALUE = MSG_ERR_PLUGIN }

Represents the values of error codes.

- enum _MSG_SESSION_PRIORITY_E {

  MSG_SESSION_PRIORITY_LOWEST, MSG_SESSION_PRIORITY_LOW, MSG_SESSION_PRIORITY_NORMAL, MSG_SESSION_PRIORITY_HIGH,

  MSG_SESSION_PRIORITY_HIGHEST }

  Represents the values of session priority.

- enum _MSG_MESSAGE_PRIORITY_E { MSG_MESSAGE_PRIORITY_LOW, MSG_MESSAGE_PRIORITY_NORMAL, MSG_MESSAGE_PRIORITY_HIGH }

  Represents the values of message priority.

- enum _MSG_STATUS_E {

  MSG_STATUS_UNKNOWN = 0, MSG_STATUS_READ, MSG_STATUS_UNREAD, MSG_STATUS_REPLIED,

  MSG_STATUS_FORWARDED, MSG_STATUS_DRAFT, MSG_STATUS_EDITING, MSG_STATUS_PROTECTED,

  MSG_STATUS_SENDING, MSG_STATUS_SENT, MSG_STATUS_FAIL_TO_SEND, MSG_STATUS_SAVED_NOTSENT,

  MSG_STATUS_SENT_NOTSAVED, MSG_STATUS_SENT_SAVED, MSG_STATUS_FAIL_TO_SENDSAVE, MSG_STATUS_REQ_CANCELLED,

  MSG_STATUS_USER_DEF_BEGIN = 0x08000000 }

  Represents the values of status.

- enum _MSG_REQUEST_TYPE_E {

  MSG_REQUEST_TYPE_INVALID = 0, MSG_REQUEST_TYPE_RESERVED_BEGIN, MSG_REQUEST_TYPE_SEND, MSG_REQUEST_TYPE_SENDSAVE,

  MSG_REQUEST_TYPE_RESERVED_END = MSG_REQUEST_TYPE_RESERVED_BEGIN + 0x00001000, MSG_REQUEST_TYPE_USER_DEF_BEGIN = 0x08000000 }

Represents the values of request types.

- enum _MSG_STORAGE_ID_VALUE_E { MSG_STORAGE_UNKNOWN = 0, MSG_STORAGE_DEFAULT }
- enum _MSG_MEDIA_TYPE_E { MSG_MEDIA_FLASH, MSG_MEDIA_REMOVABLE, MSG_MEDIA_SIM }

  Represents the values of media types.

- enum _MSG_FOLDER_TYPE_E {

  MSG_FOLDER_TYPE_ROOT, MSG_FOLDER_TYPE_INBOX, MSG_FOLDER_TYPE_OUTBOX, MSG_FOLDER_TYPE_DRAFTS,

  MSG_FOLDER_TYPE_SENT, MSG_FOLDER_TYPE_DELETED, MSG_FOLDER_TYPE_USERDEFINE }

  Represents the values of folder types.

- enum _MSG_CONTACT_TYPE_E { MSG_CONTACT_TYPE_UNKNOWN = 0, MSG_CONTACT_TYPE_EMAIL, MSG_CONTACT_TYPE_PLMN }

  Represents the values of Contact types.

- enum _MSG_ENCODE_TYPE_E {

  MSG_ENCODE_UTF8, MSG_ENCODE_UTF16, MSG_ENCODE_ASCII, MSG_ENCODE_GSM7BIT,

  MSG_ENCODE_UCS2, MSG_ENCODE_SHIFT_JIS }

  Represents the values of String encoding types.

- enum _MSG_SIZE_UNIT_E {

  MSG_SIZE_UNIT_BYTE, MSG_SIZE_UNIT_KBYTE, MSG_SIZE_UNIT_MBYTE, MSG_SIZE_UNIT_GBYTE,

  MSG_SIZE_UNIT_NUMBER }

6.16.2 Define Documentation

6.16.2.1 #define bool unsigned char

Defines bool type.

6.16.2.2 #define false 0

Defines false value.

6.16.2.3 #define MSG_DEFAULT_ACCOUNT_ID 0

Defines default account id.

6.16.2.4 #define MSG_DEFAULT_FOLDER_ID 0

Defines default folder id.

6.16.2.5 #define MSG_ERROR_BEGIN (-255)

Defines begin value for error.

6.16.2.6 #define MSG_INVALID_ACCOUNT_ID 0xffff

Defines invalid id for account.

6.16.2.7 #define MSG_INVALID_FOLDER_ID 0xffff

Defines invalid id for folder.

6.16.2.8 #define MSG_INVALID_MESSAGE_ID 0xffffffff

Defines invalid id for message.

6.16.2.9 #define true 1

Defines true value.

6.16.3 Typedef Documentation

6.16.3.1 typedef struct _MSG_ACCOUNT_ID_LIST_S MSG_ACCOUNT_ID_LIST_S

6.16.3.2 typedef unsigned int MSG_ACCOUNT_ID_T

Represents an account id.

6.16.3.3 typedef int MSG_CONTACT_TYPE_T

Represents a contact type.

The values eor this type SHOULD be in _MSG_CONTACT_TYPE_E

6.16.3.4 typedef int MSG_ENCODE_TYPE_T

Represents an encoding type.

The values for this type SHOULD be in _MSG_ENCODE_TYPE_E

6.16.3.5 typedef int MSG_ERROR_T

Represents an error code.

The values for this type SHOULD be in _MSG_ERROR_E

6.16.3.6 typedef struct _MSG_FOLDER_ID_LIST_S MSG_FOLDER_ID_LIST_S

6.16.3.7 typedef unsigned int MSG_FOLDER_ID_T

Represents a folder id.

6.16.3.8 typedef unsigned int MSG_FOLDER_TYPE_T

Represents a folder type.

The values for this type SHOULD be in _MSG_FOLDER_TYPE_E

6.16.3.9 typedef int MSG_MAIN_TYPE_T

Represents a message main type.

The values for this type SHOULD be in _MSG_MAIN_TYPE_E

6.16.3.10 typedef unsigned int MSG_MEDIA_TYPE_T

Represents a media type.

The values for this type SHOULD be in _MSG_MEDIA_TYPE_E

6.16.3.11 typedef struct _MSG_MESSAGE_ID_LIST_S MSG_MESSAGE_ID_LIST_S

6.16.3.12 typedef unsigned long MSG_MESSAGE_ID_T

Represents a messsage id.

6.16.3.13 typedef int MSG_MESSAGE_PRIORITY_T

Represents a message priority.

The values for this type SHOULD be in _MSG_MESSAGE_PRIORITY_E

6.16.3.14 typedef struct _MSG_MESSAGE_TYPE_LIST_S MSG_MESSAGE_TYPE_LIST_S

6.16.3.15 typedef struct _MSG_MESSAGE_TYPE_S MSG_MESSAGE_TYPE_S

6.16.3.16 typedef unsigned int MSG_PLUGIN_ERROR_T

Error code specific for plugins.

6.16.3.17 typedef unsigned int MSG_REQUEST_ID_T

Represents a request id, which is unique for each request.

For details of request, please refer to Request .

6.16.3.18 typedef int MSG_REQUEST_TYPE_T

Represents a request type.

The values for this type SHOULD be in _MSG_REQUEST_TYPE_E

6.16.3.19 typedef int MSG_SESSION_PRIORITY_T

Represents a session priority.

The values for this type SHOULD be in _MSG_SESSION_PRIORITY_E

6.16.3.20 typedef int MSG_SIZE_UNIT_T

Represents the unit of the size.

The values for this type SHOULD be in _MSG_SIZE_UNIT_E

6.16.3.21 typedef int MSG_STATUS_T

Represents a status.

The values for this type SHOULD be in _MSG_STATUS_E

6.16.3.22 typedef struct _MSG_STORAGE_ID_LIST_S MSG_STORAGE_ID_LIST_S

6.16.3.23 typedef unsigned int MSG_STORAGE_ID_T

Represents a Storage id.

For details please refer to Storage .

6.16.3.24 typedef struct _MSG_STRING_S MSG_STRING_S

6.16.3.25 typedef int MSG_SUB_TYPE_T

Represents a message sub type.

Each main type message can be divided into some sub types. For instance of SMS, the sub types can be CLASS0 SMS, CLASS2 SMS and so on.

6.16.4 Enumeration Type Documentation

6.16.4.1 enum _MSG_CONTACT_TYPE_E

Represents the values of Contact types.

When MSG_CONTACT_TYPE_EMAIL is used, the contact value is an email address; When MSG_-CONTACT_TYPE_PLMN is used, the contact value is a mobile number.

Enumerator:

MSG_CONTACT_TYPE_UNKNOWN The contact type is unknown

MSG_CONTACT_TYPE_EMAIL The contact type is for email address like abc@example.comemail

MSG_CONTACT_TYPE_PLMN The contact type is for phone number like +1012345678

6.16.4.2 enum _MSG_ENCODE_TYPE_E

Represents the values of String encoding types.

Enumerator:

MSG_ENCODE_UTF8 The string encoding type is UTF8

MSG_ENCODE_UTF16 The string encoding type is UTF16

MSG_ENCODE_ASCII The string encoding type is ASCII

MSG_ENCODE_GSM7BIT The string encoding type is GSM7BIT

MSG_ENCODE_UCS2 The string encoding type is UCS2

MSG_ENCODE_SHIFT_JIS The string encoding type is Shift JIS

6.16.4.3 enum _MSG_ERROR_E

Represents the values of error codes.

Success code is zero, all error codes SHOULD be negative and smaller than MSG_ERROR_BEGIN

Enumerator:

MSG_SUCCESS Successful

MSG_ERR_FS_ERROR File system error

MSG_ERR_FS_FULL File system is full

MSG_ERR_INVALID_STORAGE Storage id is invalid

MSG_ERR_INVALID_MSG_TYPE Message type is invalid

MSG_ERR_INVALID_SORT_TYPE Sort type is invalid

MSG_ERR_INVALID_FOLDER_ID Folder Id is invalid

MSG_ERR_INVALID_STORAGE_REG Storage registry is invalid

MSG_ERR_INVALID_CONNECTION Connection is invalid

MSG_ERR_NULL_POINTER Pointer is NULL

MSG_ERR_INVALID_MESSAGE_ID Message id is invalid

MSG_ERR_INVALID_ACCOUNT_ID Account id is invalid

MSG_ERR_INVALID_QUERY_ID Query id is invalid

MSG_ERR_SESSION_NOT_CONNECTED Session is not connected

MSG_ERR_INVALID_SESSION_HANDLE Session handle is invalid

MSG_ERR_PERMISSION_DENIED Permisssion is denied

MSG_ERR_STORAGE_NOT_INIT Storage is not initialized

MSG_ERR_STORAGE_REG_NOT_INIT Storage registry is not initialized

MSG_ERR_INVALID_SESSION_PRIORITY Session priority is invalid

MSG_ERR_INVALID_FILTER_ID Filter Id is invalid

MSG_ERR_INVALID_REQUEST Request is invalid: Request type is unsupported; The message type in request is invalid.

MSG_ERR_INVALID_FOLDER_TYPE Folder type is invalid

MSG_ERR_INVALID_FOLDER Folder information is invalid

MSG_ERR_NOT_EMPTY_FOLDER Folder is not empty

MSG_ERR_INVALID_ACCOUNT Account information is invalid

MSG_ERR_CONNECTION_NOT_CONNECT Connection is not connected

MSG_ERR_INVALID_MESSAGE Connection is not connected

MSG_ERR_STORAGE_FULL Storage is full

MSG_ERR_ROOTFOLDER_INUSE Root folder is associated with account(s)

MSG_ERR_STORAGE_ERROR Storage is error

MSG_ERR_MEMORY_ERROR Memory is error

MSG_ERR_COMMUNICATION_ERROR Communication between client and server is error

MSG_ERR_MAX_NUMBER_REACHED Max number reached when adding message,account or folder

MSG_ERR_INVALID_REQUEST_ID Request Id is invalid

MSG_ERR_UNKNOWN Unknown errors

MSG_ERR_FRAMEWORK_RESERVE_BEGIN Framework reserved error code begin value

MSG_ERR_FRAMEWORK_RESERVE_END Framework reserved error code end value

MSG_ERR_PLUGIN Generic error code for plugin

MSG_ERR_PLUGIN_DEF_BEGIN_VALUE Plugin defined error code begin value. It's equal to MSG_ERR_PLUGIN

6.16.4.4 enum _MSG_FOLDER_TYPE_E

Represents the values of folder types.

Enumerator:

MSG_FOLDER_TYPE_ROOT Root folder

MSG_FOLDER_TYPE_INBOX InBox folder

MSG_FOLDER_TYPE_OUTBOX OutBox folder

MSG_FOLDER_TYPE_DRAFTS Draft folder

MSG_FOLDER_TYPE_SENT Sent folder

MSG_FOLDER_TYPE_DELETED Trash folder

MSG_FOLDER_TYPE_USERDEFINE Folder is created by user

6.16.4.5 enum _MSG_MAIN_TYPE_E

Represents the values of message main types.

Four message main types are predefined: SMS, MMS, Email and iMode. More main message types can be defined here.

Enumerator:

MSG_MESSAGE_TYPE_UNKNOWN Unknown message type

MSG_MESSAGE_TYPE_MMS MMS message

MSG_MESSAGE_TYPE_EMAIL Email message

MSG_MESSAGE_TYPE_SMS SMS message

MSG_MESSAGE_TYPE_IMODE i-mode message

6.16.4.6 enum _MSG_MEDIA_TYPE_E

Represents the values of media types.

Enumerator:

MSG_MEDIA_FLASH Media type is flash

MSG_MEDIA_REMOVABLE Media type is removable

MSG_MEDIA_SIM Media type is SIM card

6.16.4.7 enum _MSG_MESSAGE_PRIORITY_E

Represents the values of message priority.

Enumerator:

MSG_MESSAGE_PRIORITY_LOW Low priority

MSG_MESSAGE_PRIORITY_NORMAL Normal priority

MSG_MESSAGE_PRIORITY_HIGH High priority

6.16.4.8 enum _MSG_REQUEST_TYPE_E

Represents the values of request types.

This enum is used as the value of type MSG_REQUEST_TYPE_T. Each message plugin can define the supported request types in this enum. Application can use the request type to invoke plugin corresponding operation via framework. Note that the enum between MSG_REQUEST_TYPE_RESERVED_BEGIN and MSG_REQUEST_TYPE_RESERVED_END are reserved for framework. For details of request, please refer to Request .

Enumerator:

MSG_REQUEST_TYPE_INVALID It's used as the initial value to request type.

MSG_REQUEST_TYPE_RESERVED_BEGIN Framework reserved begin value

MSG_REQUEST_TYPE_SEND Request to send message.

This request is used to send a message. After this request is submitted, following status may be triggered:
MSG_STATUS_SENDING: Message is in sending.
MSG_STATUS_SENT: Messag is sent successfully.
MSG_STATUS_FAIL_TO_SEND: Message is fail to send.
If the request is cancelled successfully, the message is not sent and following status is triggered:
MSG_STATUS_REQ_CANCELLED: Request is cancelled.

MSG_REQUEST_TYPE_SENDSAVE Request to send and save message.

This request is used to send and save a message. After this request is submitted, following status may be triggered:

MSG_STATUS_SENDING: Message is in sending.

MSG_STATUS_SAVED_NOTSENT: Messag is saved but not sent.

MSG_STATUS_SENT_NOTSAVED: Message is sent but not saved.

MSG_STATUS_SENT_SAVED: Message is both saved and sent.

MSG_STATUS_FAIL_TO_SENDSAVE: Message is neither saved nor sent.

If the request is cancelled successfully, the message is not sent and not saved. Following status is triggered:

MSG_STATUS_REQ_CANCELLED: Request is cancelled.

MSG_REQUEST_TYPE_RESERVED_END Framework reserved end value

MSG_REQUEST_TYPE_USER_DEF_BEGIN User defined request type begin value

6.16.4.9 enum _MSG_SESSION_PRIORITY_E

Represents the values of session priority.

Enumerator:

MSG_SESSION_PRIORITY_LOWEST Lowest priority

MSG_SESSION_PRIORITY_LOW Low priority

MSG_SESSION_PRIORITY_NORMAL Normal priority

MSG_SESSION_PRIORITY_HIGH High priority

MSG_SESSION_PRIORITY_HIGHEST Highest priority

6.16.4.10 enum _MSG_SIZE_UNIT_E

Enumerator:

MSG_SIZE_UNIT_BYTE Unit is byte

MSG_SIZE_UNIT_KBYTE Unit is kilo-byte

MSG_SIZE_UNIT_MBYTE Unit is mega-byte

MSG_SIZE_UNIT_GBYTE Unit is giga-byte

MSG_SIZE_UNIT_NUMBER Unit is number

6.16.4.11 enum _MSG_STATUS_E

Represents the values of status.

This enum is used as the value of type MSG_STATUS_T. The specific status for each message can be defined in this enum. For instance of Email, sending status can be defined to track the email sending progress.

Enumerator:

MSG_STATUS_UNKNOWN Unknown status

MSG_STATUS_READ Message is read

MSG_STATUS_UNREAD Message is unread

MSG_STATUS_REPLIED Message is replied

MSG_STATUS_FORWARDED Message is forwarded

MSG_STATUS_DRAFT Message is draft

MSG_STATUS_EDITING Message is in editing

MSG_STATUS_PROTECTED Message is protected

MSG_STATUS_SENDING Message is sending

MSG_STATUS_SENT Message is sent

MSG_STATUS_FAIL_TO_SEND Message is fail to send

MSG_STATUS_SAVED_NOTSENT Message is saved, but not sent

MSG_STATUS_SENT_NOTSAVED Message is sent, but not saved

MSG_STATUS_SENT_SAVED Message is sent and saved

MSG_STATUS_FAIL_TO_SENDSAVE Message is not saved and not sent

MSG_STATUS_REQ_CANCELLED Request is cancelled

MSG_STATUS_USER_DEF_BEGIN User defined status begin value

6.16.4.12 enum _MSG_STORAGE_ID_VALUE_E

Enumerator:

MSG_STORAGE_UNKNOWN

MSG_STORAGE_DEFAULT The default storage

Chapter 7

Page Documentation

7.1 Overview of the LiMo Messaging API

The following sections describe the Common structures and MAPI Functions the LiMo Messaging API provides to the application. In addition, the concepts of a Session and Request are discussed.

7.1.1 Common structures

In MAPI, there are few main concepts of data:

- Message
- Folder
- Account

7.1.1.1 Message

In LiMo messaging, a message represents data for a SMS, MMS, e-mail or i-mode message. A message is not specific to any one service, but supports the extension of this structure. For service-specific messages, please refer to Extendability.

Each message has a unique `messageId` in the entire messaging framework. The message information can be retrieved from the storage by its `messageId`.

For each message, there is one account attached to it. The concept is embodied on there is `accountId` field in _MSG_MESSAGE_S. If `storageId` field is not set, then the `defaultStorageId` in attached account would be used. For more information on account, please refer to Account

This concept is defined in Msg_Mapi_Message.h

7.1.1.2 Folder

A folder represents a container of messages or other folders. It should have only one parent folder, and zero or more children.

Each folder has a unique `folderId` in the entire messaging framework. The folder information can be retrieved from the storage by its `folderId`.

A folder can specify where it is stored. The storage where a folder is stored can be different from the storage where its containing messages are stored.

One of special folders is called "root folder", which is top-level folder and contains five predefined folders, which are:

- INBOX,
- OUTBOX,
- DRAFTS,
- SENT, and
- DELETED

The predefined folders cannot be created or deleted individually, instead they can be created and deleted during the corrsponding operations on its root folder.

This concept is defined in Msg_Mapi_Folder.h

7.1.1.3 Account

An account represents the personal setting information for a message type. For example, an e-mail account can include such information as e-mail address, and SMS can include message center number.

Each account has a unique `accountId` in the entire messaging framework. The account information can be retrieved from the storage by its `accountId`. The account id is the reference being used by other types of structure, such as message and folder.

A message type can be associated with more than one accounts, and has one default account. However, one message has only one account. If application has not selected any account, then the default account takes effect. An account has an attribute called `rootFolder`, which defines the root folder having association with that account. As name suggested, `rootFolder` field needs to be set with root folder only, other folder cannot be set. A folder could be shared among different accounts.

This concept is defined in Msg_Mapi_Account.h

7.1.2 Session, Request and Filter Rules

7.1.2.1 Session

Session is the connection between application and MAPI for message reception and transmission.

When an application wants to send or receive a message, it must create and connect a session first. A session can be assigned with following information:

- Register MsgOnMessageIncomingCallback function.
- Register MsgOnStatusChangedCallback function.
- Set priority. For details please refer to MsgSetPriority
- Set filter rule.

One application can create and connect more than one sessions, different sessions can be assigned with different priorities and filter rules.

To know how to use asynchronous callback for communication between application and framework, please refer to Callback Functions

7.1.2.2 Request

A request contains the information that is used to command framework for services, such as sending a message. A request is sent to Messaging Framework via a session, and Messaging Framework chooses proper plugin to handle the request.

A common usage for the request is to send a message. And the request type is predefined as MSG_-REQUEST_TYPE_SEND.

This concept is defined in Msg_Mapi_Message_Request.h

7.1.2.3 Callback Functions

Callback is used by application to detect incoming message and underlayer status change. Two callback functions prototype are provided:

- one is MsgOnMessageIncomingCallback, which is to listen incoming message,
- the other is MsgOnStatusChangedCallback, which is to listen underlayer status change.

Both callback functions are implemented by applications and should be bound to a session.

7.1.2.4 Filter and Filter List

When an application is only interested in some kinds of messages, filter list is used to filter these specific messages and return them to the application.

Noted that filter is for filtering incoming messages, not a storage concept. For storage, the filter concept is called Query.

Filter is the basic unit to set the rules, it tells the filter service how to filter the message. Filter list is composed of various filters. They are defined in Msg_Mapi_FilterRule.h .

Application can set a filter list for a session to get the interesting message. Please refer to _MSG_FILTER_S and _MSG_FILTER_LIST_S for details.

MAPI does NOT define any filters. Application developers should refer to the specific implementation of filter service to get the supported filters.

7.1.3 Storage

7.1.3.1 Connection

A connection is a logical concept between client application and LiMo Messaing framework. It does NOT represent the physical storage connection underneath the MAPI. A connection is required by the storage functions, such as adding message, as their first parameter.

There are found status in connection's life-cycle:

- created(but-not-yet-connected),
- connected,
- disconnected and
- destroyed.

The following functions are responsible to change life-cycle status:

- MsgCreateStoConnection
- MsgOpenStoConnection
- MsgCloseStoConnection
- MsgDestroyStoConnection

For more information, please refer to Msg_Mapi_Storage_Manager.h

7.1.3.2 Storage

Storage is a collection of messages and related information that are stored on one storage media. Storage media is the physical data storage device, such as SD card and Flash. One storage media can have more than one storages.

For example, on flash storage media, there are two storages which are located at two different folders: /flash/Folder1/ and /flash/Folder2/ respectively. Since storage media can be removable, the storages on the media can be detached and attached dynamically. For instance, user plugs out SD card from UE 1 and plugs in this card into UE 2, all the messages stored in this card will be hidden from UE 1 and be visible to UE 2.

For details information, please refer to _MSG_STORAGE_REGISTRY_S

Application can query supported storages by MAPI method MsgListStorageRegistry and MsgGetStorageRegistry.

For sample code, please refer to Storage General: Initializing Storage

7.1.3.3 Query

LiMo Messaging supports query on message retrieval from storage.

Query rule, represented by _MSG_QUERY_S, is a rule to specify a single search criteria for messages. Query rule list, represented by _MSG_QUERY_LIST_S, is composed of a number of query rules. If one message satisfies one of query rule in the list, it satisfies the query rule list.

For more information on how to use Query, please refer to MsgQueryMessage .

7.1.3.4 Sort

LiMo Messaging supports sort on message retrieval from storage.

Sort rule, represented by _MSG_SORT_RULE_S, defines which field in message structure is sorted on, and how this field is sorted.

Sort type, represented by _MSG_SORT_TYPE_T, defines which field can be sorted on.

For more information on how to use Sort, please refer to MsgQueryMessage .

7.1.4 Extendability

There are several structures in LiMo Messageing designed to be extended. They are:

- _MSG_ACCOUNT_S
- _MSG_FOLDER_S
- _MSG_MESSAGE_S
- _MSG_MESSAGE_REQUEST_S

In these structures, there are two fields for data extension, which are :

- `pExtData`
- `extDataSize`

`pExtData`, of type `void *`, can hold a pointer to any structure. Noted that the pointed structure cannot contain any pointer as its field, since the extended data will be treated as a memory block to pass around between applications and plugins, any inside pointer would not be handled properly.

`extDataSize`, of type size_t, indicates the length of the `pExtData` block.

7.1.5 MAPI Functions

This section describes the functions that MAPI provides.

7.1.5.1 Message Sending and Receiving

The Message sending and receiving module provides the unique interfaces to send and receive messages. The Session structure is the primary structure to hold the connection information. Applications use Session to send and receive messages.

The major functionality that the message sending and reception module provides are:

- Creating and destroying session handle: MsgCreateSession and MsgDestroySession, respectively.
- Connecting and disconnecting session handle: MsgConnectSession and MsgDisconnectSession .
- Defining the callback functions prototypes: on message received and on status changed. See MsgOnMessageIncomingCallback and MsgOnStatusChangedCallback.
- Registering the callback functions on session handle: MsgRegMessageCallback and MsgRegStatusCallback.
- Setting priority and filter list on session handle. See MsgSetPriority and MsgSetFilterList.
- Submitting and cancelling a request; MsgSubmitReq and MsgCancelReq.

The following is the file for this module:

- Msg_Mapi_Session.h

For more information, please see the sample code:

- Sending a Message
- Receiving Messages

7.1.6 Storage Functionality

7.1.6.1 Storage Management

The Storage management module provides the management functionality for storage such as storage registry operations, storage data initialization etc. The major functions that the Storage Management module provides are:

- Listing and getting storage registration information. See MsgListSortType and MsgGetStorageRegistry.
- Creating, opening, closing, and destroying connection; MsgCreateStoConnection, MsgOpenStoConnection, MsgCloseStoConnection, and MsgDestroyStoConnection, respectively.
- Getting Framework Storage and Message Storage status. See MsgGetFwStorageStatus and MsgGetMsgStorageStatus.
- Releasing memory of storage id list and storage registry information. See MsgReleaseStorageIdList and MsgReleaseStorageRegistry. The following is the file for this module:
- Msg_Mapi_Storage_Manager.h

For more information, please see the sample code:

- Initializing Storage

7.1.6.2 Message Storage

The major functions that the message storage module provides are:

- Adding and updating messages; MsgAddMessage and MsgUpdateMessage.
- Counting messages and counting unread messages. See MsgCountMessage and MsgCountUnreadMessage.
- Moving messages. See MsgMoveMessageToFolder and MsgMoveMessageToStorage.
- Querying messages: MsgQueryMessage.
- Getting message ids by storage. See MsgGetMessageIdsByStorage.
- Getting message detail information, getting message common information and getting message parent folder. Refer to MsgGetMessage, MsgGetMessageCommInfo and MsgGetMessageParentId.
- Releasing memory of message detail information (MsgReleaseMessage) and message common information (MsgReleaseMessageCommInfo).

The functionality is defined in:

- Msg_Mapi_Storage_Message.h.

For more information, please see the sample code:

- Creating and Storing a Message
- Retrieving Message List and Entity

7.1.6.3 Folder Storage

The major functionality that folder storage module provides:

- Adding and updating folders: MsgAddFolder and MsgUpdateMessage.
- Counting and querying folders: MsgCountSubFolder and MsgQueryFolder.
- Moving folders. Please see MsgMoveFolderToFolder and MsgMoveFolderToStorage
- Getting folder ids by storage and getting all root folder ids. See MsgGetFolderIdsByStorage and MsgGetAllRootFolderIds.
- Getting the specified folder (MsgGetFolder) and getting parent folder id (MsgGetFolderParentId).
- Releasing memory of folder: MsgReleaseFolder.

The functionality is defined in:

- Msg_Mapi_Storage_Folder.h

For more information, please see the sample code:

- Creating and Storing a Folder
- Querying Folders

7.1.6.4 Account Storage

The major functionality that the Account storage module provides:

- Adding and updating. See MsgAddAccount and MsgUpdateAccount.
- Counting (MsgCountAccount) and querying (MsgQueryAccount) accounts in storage.
- Moving accounts. See MsgMoveAccountToStorage.
- Getting account ids by storage. See MsgGetAccountIdsByStorage.
- Getting the specified account in storage: MsgGetAccount.
- Getting and setting default account id: MsgGetDefaultAccountID and MsgSetDefaultAccountID.
- Getting account common information: MsgGetCommAccount.
- Releasing the memory of account information (MsgReleaseAccount) and account common information (MsgReleaseCommAccount).

The functionality is defined in:

- Msg_Mapi_Storage_Account.h

For more information, please see the sample code:

- Creating and Storing an Account
- Updating an Account
- Getting a Default Account

7.2 Sample Code for Using the LiMo Messaging API

The following samples of application code demonstrate many of the tasks an application programmer will need to perform.

- Extending a Message Structure
- Sending a Message
- Receiving Messages
- Initializing Storage
- Creating and Storing a Message
- Retrieving Message List and Entity
- Creating and Storing an Account
- Updating an Account
- Getting a Default Account
- Creating and Storing a Folder
- Querying Folders

7.3 Extending a Message Structure

Since LiMo MAPI provides only a generic message, the sample code below demonstrates how to make SMS a specific message service. This requires an extended Account structure as defined below. All the extended structures should be shared between applications and plug-ins. Plug-in developers should implement all the plug-in interface functions based on these extended structures, and application developers can use these extended structures in their own application.

MSG_ERR_USER_DEFINED represents an application-defined error code. Applications should define their own error codes in their implementations.

```
typedef struct _MSG_MESSAGE_SMS_EXTDATA_S
{
    char * pBody; //Body
    char * pScAddr; //Service Center
    int expirePeriod; //expirePeriod
}_MSG_MESSAGE_SMS_EXTDATA_S;

// This structure is the extended structure for _MSG_ACCOUNT_S,
// which can be used as Email account structure.
typedef struct _MSG_ACCOUNT_EMAIL_EXTDATA_S
{
    _MSG_STRING_S emailServerUrl;
    int portNumber;
    _MSG_STRING_S displayAddress;
} MSG_ACCOUNT_EMAIL_EXTDATA_S;

// This structure is the extended structure for _MSG_ACCOUNT_S,
// which can be used as SMS account structure.
typedef struct _MSG_ACCOUNT_SMS_EXTDATA_S
{
    _MSG_STRING_S smsServerCenter;
    int portNumber;
    bool needDeliveryReport;
    bool autoReply;
} _MSG_ACCOUNT_SMS_EXTDATA_S;

//The sub type enum of SMS
enum _MSG_SMS_SUB_TYPE_E
{
    MSG_TYPE_SMS_TEXT
};

//define the storage id enum
enum _MSG_STORAGE_ID_E
{
    MSG_STORAGE_ID_FLASH,
    MSG_STORAGE_ID_SD
};
```

7.4 Message Session: Sending a Message

In this section, you will learn how to send a message through LiMo MAPI. This task requires two pieces of structure to be prepared. One is the message structure, composed in the application The other is a Session structure. A connected Session is required for both sending a message, and receiving a message as will be demonstrated.

This task is considered asynchronous, that means after the application sends the sending request, the event of request's status will be returned afterwards through a registered callback function.

To send a SMS, there is one precondition: there should be a SMS plug-in already plugged into the framework. The SMS plug-in should provide SMS extension structure and subtypes for SMS. To get this result, the application MUST implement the Callback function MsgOnStatusChangedCallback(), and register it to the session.

Suppose the structure and subtype are defined as in the previous example. The callback function is defined as shown below.

```
void MyOnStatusChange(_MSG_MESSAGE_STATUS_S * ppMsgStatus, MSG_SESSION_HANDLE_T hSession)
{
    //Get the request id and status from ppMsgStatus here.
}

int reqId;
MSG_ERROR_T SendMessageSample()
{
    MSG_SESSION_HANDLE_T Session;
    MSG_MESSAGE_REQUEST_S req;
    MSG_ERROR_T err;
    _MSG_MESSAGE_S msgInfo; //Message to be sent
    _MSG_CONTACT_S to[2]; //contact array which are used by tolist
    _MSG_CONTACT_LIST_S tolist; // to whom the message to be sent
    _MSG_MESSAGE_SMS_EXTDATA_S SMSInfo; // SMS extension info
    MSG_STRING_S strSubject;

    err = MsgCreateSession(&Session); // Create session
    if(err != MSG_SUCCESS) //if creation failed, return error.
        return err;

    err = MsgConnectSession(Session); //Establish Session Connection.
    if(err != MSG_SUCCESS) //if connection failed, destroy SessionHandle, return error.
    {
        MsgDestroySession(Session);
        return err;
    }

    // Register callback function
    err = MsgRegStatusCallback(Session, MyOnStatusChange);
    if(err != MSG_SUCCESS)
    {
        MsgDisconnectSession(Session);
        MsgDestroySession(Session);
        return err;
    }

    //Set To
    to[0].contactType = MSG_CONTACT_TYPE_PLMN;
    to[0].contactVal = "000"; //1st phone number to be sent. It is just a sample number.
    to[1].contactType = MSG_CONTACT_TYPE_PLMN;
    to[1].contactVal = "000"; //2nd phone number to be sent. It is just a sample number.
    tolist.pContacts = to; //fill tolist
    tolist.iCount = 2;
    msgInfo.to = tolist; //set tolist to msgInfo

    //Set Subject
```

```
    strSubject.encodeType = MSG_ENCODE_ASCII;
    strSubject.pString = "subject";
    strSubject.length = strlen(strSubject.pString);
    msgInfo.subject = strSubject;

    //set msgType
    msgInfo.msgType.mainType = MSG_MESSAGE_TYPE_SMS;
    msgInfo.msgType.subType = MSG_TYPE_SMS_TEXT;

    //set user defined ext data, MAPI user can read SMS plug-in document
    // to understand the meaning of each
    SMSInfo.pScAddr = "222"; //it is just a sample number.
    SMSInfo.pBody = "This is a test SMS";
    SMSInfo.expirePeriod = 0;
    msgInfo.pExtData = &SMSInfo; //Sent extension data to msgInfo.

    // Get the accountId used to send message and set it to msgInfo.accountId.
    // For details of account information,

    //fill req
    req.reqType = MSG_REQUEST_TYPE_SEND; // Set request type
    req.pMsg = &msgInfo; // Set msg
    err = MsgSubmitReq(Session, &req, &reqId); // Submit request.


    // finalize session.
    // This is just a sample to show how to finalize session.
    // Usually, application can keep a connected session
    MsgDisconnectSession(Session);
    MsgDestroySession(Session);

    return err;

}
```

7.5 Message Session: Receiving Messages

In this section, you will learn how to receive incoming message in your application, through LiMo MAPI. In this example, only SMS messages will received and processed. Other r types of messages will be ignored by the application.

To receive messages, an application MUST implement the callback function MsgOnMessageIncoming-Callback() and register it to session. Beyond that, an application MAY set a filter list. If a filter list is set, only incoming messages that satisfy the filter list will be passed to the registered callback function. In the code below, MyOnMsgIncome() is the callback function that implements MsgOnMessageIncomingCall-back().

```
void MyOnMsgIncome(_MSG_MESSAGE_S * ppMsgInfo,
                   _MSG_MESSAGE_STATUS_S * ppMsgStatus,
                   MSG_SESSION_HANDLE_T hSession)
{
    switch(ppMsgInfo->msgType.mainType)
    {
        case MSG_MESSAGE_TYPE_SMS:
        // Handle SMS message here
        break;
        case MSG_MESSAGE_TYPE_MMS:
        // Handle MMS message here
        break;
        default:
        // Handle other message here
        break;
    }
}

MSG_ERROR_T ReceiveMessageSample()
{
    MSG_SESSION_HANDLE_T Session;
    MSG_ERROR_T err;
    _MSG_FILTER_S filters[2]; // filter items used by filter list
    _MSG_FILTER_LIST_S filterList; // filter list used by filter rule


    err = MsgCreateSession(&Session); // Create session
    if(err != MSG_SUCCESS) //if creation failed, return error.
        return err;

    err = MsgConnectSession(Session); //Establish Session Connection.
    if(err != MSG_SUCCESS) // if connection failed, destroy SessionHandle, return error.
    {
        MsgDestroySession(Session);
        return err;
    }

    // Register callback function
    err = MsgRegMessageCallback(Session, MyOnMsgIncome);
    if(err != MSG_SUCCESS)
    {
        MsgDisconnectSession(Session);
        MsgDestroySession(Session);
        return err;
    }
    // set filters
    //filters[0] is to filter all sms messages.
    filters[0].filterId = MSG_FILTER_BY_MSG_TYPE;
    filters[0].filterValue = (unsigned char * )malloc(sizeof (unsigned char));
 filters[0].filterValue = MSG_MESSAGE_TYPE_SMS;
    filters[0].valueLength = 1;
    //true means if message satisfies the filter, framework will save it automatically.
    filters[0].autoStore = true;
```

```
    // true means if message satisfies the filter, it will be received
    filters[0].intercept = true;

    //filters[1] is to filter all messages with attachment.
    filters[1].filterId = MSG_FILTER_BY_HAS_ATTACHMENT;
    filters[1].filterValue = (unsigned char * )malloc(sizeof (unsigned char));
 filters[1].filterValue = 1;
    filters[1].valueLength = 1;
    filters[1].autoStore = false;
    filters[1].intercept = false;

    // Set filterList, all the filters in filterlist are OR related,
    // former filters will be compared first.
    filterList.filters = filters;
    filterList.iCount = 2;

    // Set the filterRule to session
    MsgSetFilterList(Session, &filterList);
    if(err != MSG_SUCCESS)
    {
        MsgDisconnectSession(Session);
        MsgDestroySession(Session);
        return err;
    }

    // Set session priority
    MsgSetPriority(Session, MSG_SESSION_PRIORITY_NORMAL);
    if(err != MSG_SUCCESS)
    {
        MsgDisconnectSession(Session);
        MsgDestroySession(Session);
        return err;
    }
}
```

7.6 Storage General: Initializing Storage

For message, folder and account storage, storage has to be connected prior to use. The framework is responsible for storage setup and initialization. The application has to connect to the framework, before operating storage.

The following code illustrates how to connect and disconnect the storage connection between applications and framework.

```
MSG_CONNECTION_HANDLE_T hConnection;

MSG_ERROR_T InitDB()
{
    MSG_ERROR_T ret;

    ret = MsgCreateStoConnection(&hConnection);

    if(ret != false)
    ret = MsgOpenStoConnection(hConnection);
    return ret;
}

MSG_ERROR_T Shutdown()
{
    MSG_ERROR_T ret;
    //TODO
    //.....

    ret = MsgCloseStoConnection(hConnection);

    ret = MsgDestroyStoConnection(hConnection);

    return ret;
}
```

7.7 Message Storage: Creating and Storing a Message

Pre-condition: The storage has to be properly setup and initialized. To setup a storage, please refer to the previous example.

The code first creates a SMS message structure, and then stores the structure into storage.

```
MSG_ERROR_T StoreNewSMS(MSG_CONNECTION_HANDLE_T hConnection)
{
    _MSG_MESSAGE_S msgInfo;
    _MSG_CONTACT_S contactList[2];
    _MSG_CONTACT_LIST_S to;
    _MSG_MESSAGE_REQUEST_S req;
    _MSG_MESSAGE_SMS_EXTDATA_S SMSInfo;
    _MSG_STRING_S string[2];
    _MSG_STRING_S subject;
    MSG_ACCOUNT_ID_T AccountId;
    MSG_ERROR_T ret;
    / * Set To * /
    string[0].pString = "aaa";
    string[0].length = 4;
    string[0].encodeType = MSG_ENCODE_UTF8;
    contactList[0].contactType = MSG_CONTACT_TYPE_PLMN;
    contactList[0].contactVal = "";
    contactList[0].contactName = string[0];
    contactList[0].extDataSize= 0;
    contactList[0].pExtData = NULL;
    string[1].pString = "bbb";
    string[1].length = 4;
    string[1].encodeType = MSG_ENCODE_UTF8;
    contactList[1].contactType = MSG_CONTACT_TYPE_PLMN;
    contactList[1].contactVal = "11";
    contactList[1].contactName = string[1];
    contactList[1].extDataSize= 0;
    contactList[1].pExtData = NULL;
    to.pContacts = contactList;
    to.iCount = 2;

    msgInfo.to = to;

    / * Set Subject * /
    subject.encodeType = MSG_ENCODE_UTF8;
    subject.length = 8;
    subject.pString = "subject";
    msgInfo.subject = subject;

    / * set msgType * /
    msgInfo.msgType.mainType = MSG_MESSAGE_TYPE_SMS;
    msgInfo.msgType.subType = MSG_TYPE_SMS_TEXT;

    / * set storage id * /
    msgInfo.storageId = MSG_STORAGE_ID_FLASH;

    ret = MsgGetDefaultAccountID(hConnection,msgInfo.msgType,&AccountId);
    if(ret == MSG_SUCCESS)
        msgInfo.accountId = AccountId;
    else
    return ret;

    / * set parent folder * /


    / * set priority * /

    / * set user defined ext data * /
    SMSInfo.pScAddr = "+11";
```

```
    SMSInfo.pBody = "This is a test SMS";
    SMSInfo.expirePeriod = 0;
    msgInfo.pExtData = &SMSInfo;
    msgInfo.pCommExtData = NULL;
    / * Store Message * /
    return MsgAddMessage(hConnection,&msgInfo);
}
```

7.8 Message Storage: Retrieving Message List and Entity

Pre-condition: The storage has to be properly set up and initialized. To setup a storage, please refer to the Initializing Storage example.

The code below queries a number of messages in a specific folder with query rules, and iterates the message list to handle each message.

- hConnection is a client-server connection for storage.
- folderId indicates which folder is the parent folder containing all expected messages.

```
MSG_ERROR_T QueryMsg(MSG_CONNECTION_HANDLE_T hConnection )
{
    MSG_ERROR_T ret;
    int i; // counter for result list iteration.
    MSG_MESSAGE_ID_LIST_S msgIdList;
    _MSG_QUERY_LIST_S sQueryList;
    _MSG_QUERY_S sQuery [2];
    _MSG_SORT_RULE_S SortRule;
    MSG_FOLDER_ID_T FolderId;
    _MSG_MESSAGE_S **ppMsg;
    // Create a query list.
    sQuery[0].queryId = MSG_QUERY_BY_SUBJECT;
    sQuery[0].queryValue = "Hello";
    sQuery[0].valueLength = 6;

    sQuery[1].queryId = MSG_QUERY_BY_FROM;
    sQuery[1].queryValue = "139 ******** ";
    sQuery[1].valueLength = 14;

    sQueryList.queries = sQuery;
    sQueryList.iCount = 2;
    / * set sort rule * /
    SortRule.sortType = MSG_SORT_BY_DISPLAY_TIME;
    SortRule.isAscending = true;

    / * get and set folder id * /
    FolderId = 2; / * such as: 2 represents INBOX * /

    ret = MsgQueryMessage( hConnection, FolderId,
    0, / * iBeginIndex * /
    -1, / * iCount. -1 means all records in the folder. * /
    &SortRule,
    &sQueryList,
    &msgIdList);

    if(ret != MSG_SUCCESS )
    {
        return ret;
    }
    // Iterate the result list
    i = msgIdList.iCount;
    while(i>0)
    {
        ret = MsgGetMessage( hConnection, ( * (msgIdList.msgIds))++, ppMsg);
        if(ret != MSG_SUCCESS )
        {
            / * abnormity handle * /
        }
        / * add use ppMsg code here * /

        MsgReleaseMessage( * ppMsg);
        i--;
    }
```

```
    MsgReleaseMessageIdList(&msgIdList);
    return ret;
}
```

7.9 Folder Storage: Creating and Storing a Folder

Pre-condition: The storage has to be properly setup and initialized. To setup a storage, please refer to the Initializing storage example.

The code below creates a folder under a specified parent folder.

```
//create user defined folder
MSG_ERROR_T CreateFolder(MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T parentFolderId)
{
    MSG_ERROR_T ret;
    _MSG_FOLDER_S folderInfo;
    _MSG_FOLDER_S **ppParentFolder;

    //get connection handle
    if( hConnection == NULL )
    return MSG_ERR_USER_DEFINED;

    //get parent folder
    ret = MsgGetFolder(hConnection, parentFolderId, ppParentFolder);
    if( ret != MSG_SUCCESS )
    {
        MsgReleaseFolder( * ppParentFolder);
        return MSG_ERR_USER_DEFINED;
    }

    //set the folder information which is to add,
    folderInfo.folderName.encodeType = MSG_ENCODE_ASCII;
    folderInfo.folderName.pString= "MyFolder";
    folderInfo.folderName.length = 8;
    folderInfo.folderId = MSG_INVALID_ID;
    folderInfo.parentId = parentFolderId;
    folderInfo.storageId = ( * ppParentFolder)->storageId;
    folderInfo.sortType = MSG_SORT_BY_DISPLAY_SUBJECT;
    folderInfo.folderType = MSG_FOLDER_TYPE_DELETED;
    folderInfo.commExtDataSize = 0;
    folderInfo.pCommExtData = NULL;


    //add folder
    ret = MsgAddFolder(hConnection,&folderInfo);

    MsgReleaseFolder( * ppParentFolder);

    return ret;
}
```

7.10 Folder Storage: Querying Folders

Pre-condition: The storage has to be properly setup and initialized. To setup a storage, please refer to the Initializing storage example.

The code below queries subfolders of a specific parent folder. There are two required parameters:

- hConnection represents a client-server connection for storage.
- folderId indicates which folder is the parent folder containing all expected messages. expected folders.

```
//Querying folder
MSG_ERROR_T QueryFolders(MSG_CONNECTION_HANDLE_T hConnection, MSG_FOLDER_ID_T folderId)
{
    MSG_ERROR_T ret;

    //connection handle
    if( hConnection == NULL )
    return MSG_ERR_USER_DEFINED;

    //query the direct child folders of a specific parent folder
    _MSG_FOLDER_ID_LIST_S folderList;
    ret = MsgQueryFolder(hConnection, folderId, 0, 0, &folderList);
    if( ret != MSG_SUCCESS )
    {
        MsgReleaseFolderIdList(&folderList);
        return MSG_ERR_USER_DEFINED;
    }

    // user can do something to the child folders which is just searched out.

    //free allocated memory
    MsgReleaseFolderIdList(&folderList);
    return ret;
}
```

7.11 Account Storage: Creating and Storing an Account

Pre-condition: The storage has to be properly setup and initialized. To setup a storage, please refer to the Initializing storage example. The code below is to create account. There is one required parameter:

- hConnection represents a client-server connection for storage.

```
MSG_ERROR_T CreateAccount(MSG_CONNECTION_HANDLE_T hConnection)
{
    MSG_ERROR_T ret;
    _MSG_ACCOUNT_S account;
    _MSG_FOLDER_ID_LIST_S folderList;
    MSG_STORAGE_ID_LIST_S storageIdList;

    //connection handle
    if( hConnection == NULL )
    return MSG_ERR_INVALID_CONNECTION;

    account.accountId = MSG_INVALID_ID;

    //account name and type
    account.accountName.encodeType = MSG_ENCODE_ASCII;
    account.accountName.pString = "TestAccount";
    account.accountName.length = 11;
    account.msgType.mainType = MSG_MESSAGE_TYPE_SMS;
    account.msgType.subType = MSG_TYPE_SMS_TEXT;

    //Setting the storage id.
    //For the testing purpose, the first storage id of the querying result is used as
    //the storage of account, and the second is for the default storage of message.
    MSG_STORAGE_ID_T stoId1, stoId2;
    MsgListStorageRegistry(hConnection, &storageIdList);
    if(storageIdList.iCount > 1)
    {
        stoId1 = storageIdList.pStorageIds[0];
        stoId2 = storageIdList.pStorageIds[1];
    }
    else if(storageIdList.iCount == 1)
    {
        stoId1 = storageIdList.pStorageIds[0];
        stoId2 = storageIdList.pStorageIds[0];
    }
    else
    {
        return MSG_ERR_USER_DEFINED;
    }
    account.storageId = stoId1;
    account.defaultMsgStorageId = stoId2;
    MsgReleaseStorageIdList(&storageIdList);


    //setting the root folder of this account, for the test purpose we simply use
    //the first root folder from the queried result.
    ret = MsgGetAllRootFolderIds(hConnection, &folderList);
    if( ret != MSG_SUCCESS )
    {
        return MSG_ERR_USER_DEFINED;
    }
    if(folderList.iCount > 0)
    account.rootFolderId = folderList.folderIds[0];
    else
    {
        return MSG_ERR_USER_DEFINED;
    }
    MsgReleaseFolderIdList(&folderList);
```

```
    //account extended data
    _MSG_ACCOUNT_SMS_EXTDATA_S extData;
    extData.smsServerCenter.pString = "test.motorola.com";
    extData.smsServerCenter.length = 17;
    extData.portNumber = ;
    extData.needDeliveryReport = true;
    extData.autoReply = false;
    account.pExtData = &extData;

    //Call mapi method to create account
    ret = MsgAddAccount(hConnection, &account);

    return ret;
}
```

7.12 Account Storage: Updating an Account

Pre-condition: The storage has to be properly setup and initialized. To setup a storage, please refer to the Initializing storage example.

The code below updates a specific account. The function must have a valid account id for the operation. There is one required parameter:

- hConnnection represents a client-server connection.

```
MSG_ERROR_T UpdateAccount(MSG_CONNECTION_HANDLE_T hConnection, MSG_ACCOUNT_ID_T account_id)
{
    MSG_ERROR_T ret;
    _MSG_ACCOUNT_S account;
    _MSG_ACCOUNT_S ** ppAccount_original;

    //connection handle
    if( hConnection == NULL )
        return MSG_ERR_USER_DEFINED;

    account.accountId= account_id;

    //get original account information
    MsgGetAccount(hConnection, account_id, ppAccount_original);
    //account name
    account.accountName.encodeType = MSG_ENCODE_ASCII;
    account.accountName.pString = "UpdateName";
    account.accountName.length = 10;
    //message type
    account.msgType.mainType = ( * ppAccount_original)->msgType.mainType;
    account.msgType.subType = ( * ppAccount_original)->msgType.subType;

    //storage id
    account.storageId = ( * ppAccount_original)->storageId;
    account.defaultMsgStorageId = ( * ppAccount_original)->defaultMsgStorageId;

    //account root folder list
    account.rootFolderId = ( * ppAccount_original)->rootFolderId;

    //account extended data
    _MSG_ACCOUNT_SMS_EXTDATA_S exData;
    exData.smsServerCenter.pString = "test2.motorola.com";
    exData.smsServerCenter.length = 18;
    exData.portNumber = ;
    exData.needDeliveryReport = true;
    exData.autoReply = true;
    account.pExtData = &exData;

    //Call mapi method to update account
    ret = MsgUpdateAccount(hConnection, &account);

return ret;
}
```

7.13 Account Storage: Getting a Default Account

Pre-condition: The storage has to be properly setup and initialized. To setup a storage, please refer to the Initializing storage example.

The code below gets the default account id of a SMS account.

```
MSG_ERROR_T GetDefaultAccount(MSG_CONNECTION_HANDLE_T hConnection)
{
    MSG_ERROR_T ret;

    //connection handle
    if( hConnection == NULL )
        return MSG_ERR_USER_DEFINED;

    //get the default account id
    MSG_MESSAGE_TYPE_S msgType;
    msgType.mainType = MSG_MESSAGE_TYPE_SMS;
    msgType.subType = MSG_TYPE_SMS_TEXT;


    MSG_ACCOUNT_ID_T accountId;
    ret = MsgGetDefaultAccountID(hConnection, msgType, &accountId);
    if( ret != MSG_SUCCESS )
    {
        return MSG_ERR_USER_DEFINED;
    }

    //here we can get detail information of the default account
    _MSG_ACCOUNT_S **ppAccount;
    ret = MsgGetAccount(hConnection, accountId, ppAccount);
    if( ret != MSG_SUCCESS )
    {
        MsgReleaseAccount( * ppAccount);
        return MSG_ERR_USER_DEFINED;
    }

    // we can do something here for the default account


    //destroy object and free allocated memory
    MsgReleaseAccount( * ppAccount);

    return ret;
}
```

7.14 Sequence Charts for the LiMo Messaging API

The following sequence charts are provided:

- Messaging Framework Startup
- Session: Sending a Message
- Session: Receiving Messages
- Message Storage: Creating and Opening Storage Connection
- Message Storage: Closing and Destroying Storage Connection
- Message Storage: Creating and Storing a Message
- Message Storage: Querying Messages
- Folder Storage: Creating and Storing a Folder
- Account Storage: Creating and Storing an Account
- Filter Plug-in: Set Filter List to Filter Messages
- Sending Messages (Normal Flow)
- Sending Messages (Alternative Flow)
- Receiving Messages
- Explore Message List (Flow for Listing Messages)
- Explore Message List (Flow for Browsing a Message)
- Explore Message List (Flow for Removal of a Message)
- Explore Message List (Flow for Moving Messages Between Folders)
- Manage Account and Profile (Flow for Creation of an Account)
- Manage Account and Profile (Flow for Update of an Account)
- Manage Account and Profile (Setting Default Account)
- Manage Account and Profile (Flow for Removing an Account)
- Manage Account and Profile (Alternative Flow for Removing an Account)

Legend

The chart below shows the legends used in the sequence charts.

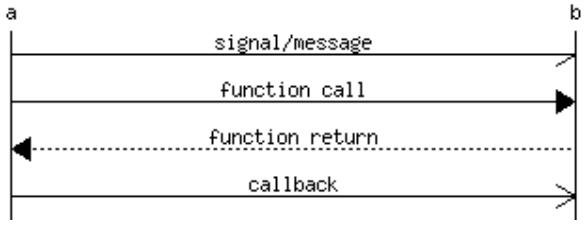

Figure 7.1: Sequence Chart Legend

Terms

- MAPI: Messaging API
- MsgFw: Messaging Framework
- TransportHandlerPlg: Transport Handler Plug-in
- FwStoPlg: Framework Storage Plug-in
- MsgStoPlg: Message Storage Plug-in
- FilterPlg: Filter plug-in

7.15 Messaging Framework Startup

Precondition: Plug-in configuration file and Storage registry configuration file are ready. Data on all storages are ready.

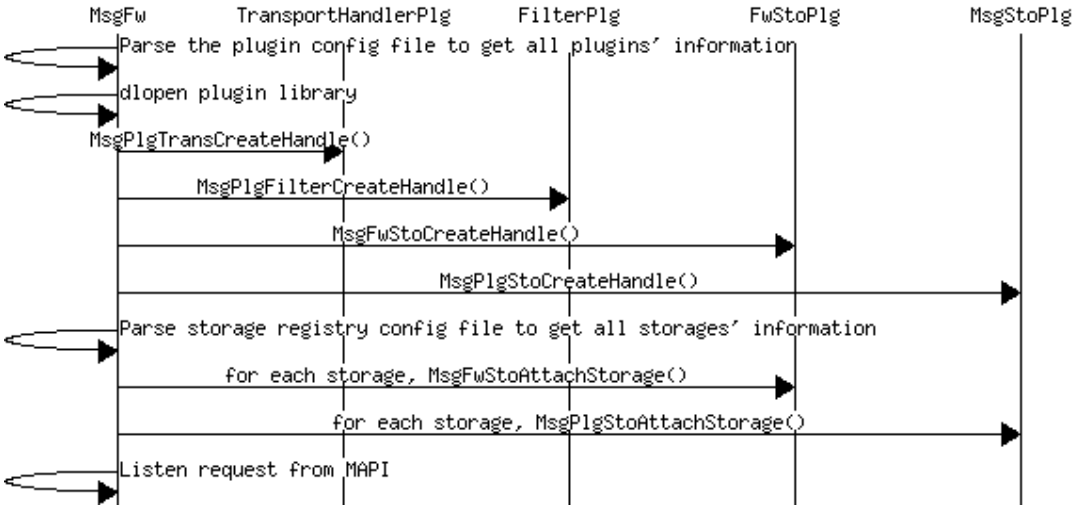

Figure 7.2: Messaging Framework Startup

7.16 Session: Sending a Message

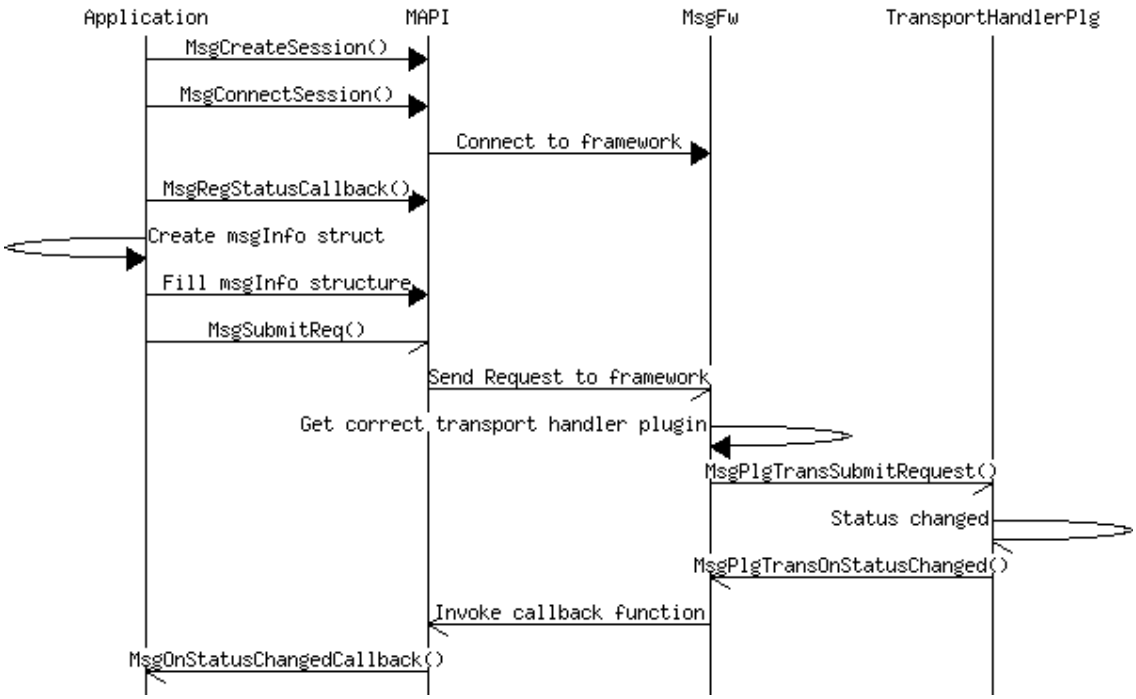

Figure 7.3: Session: Sending a Message

7.17 Session: Receiving Messages

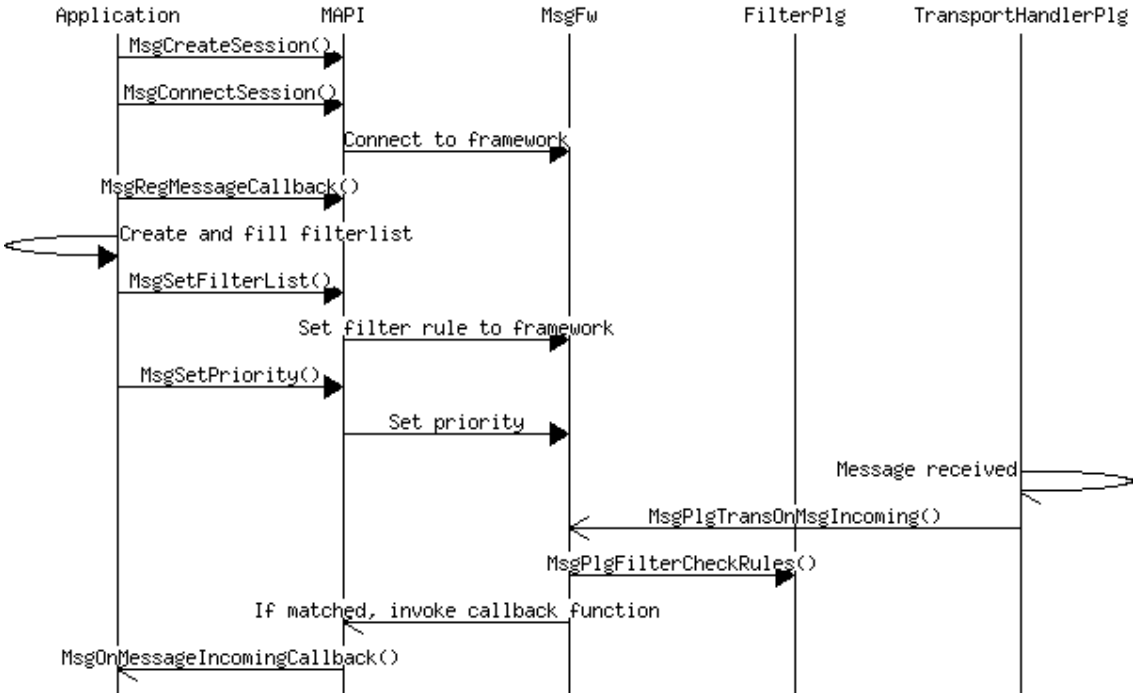

Figure 7.4: Session: Receiving Messages

7.18 Message Storage: Creating and Opening Storage Connection

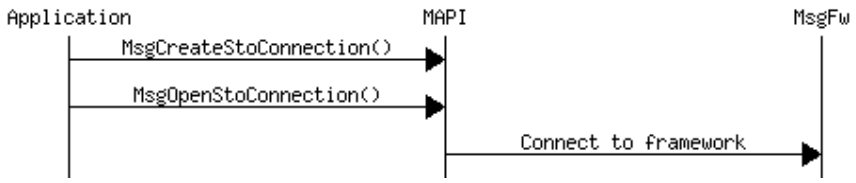

Figure 7.5: Message Storage: Creating and Opening Storage Connection

7.19 Message Storage: Closing and Destroying Storage Connection

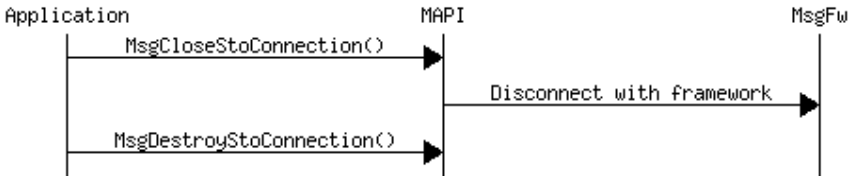

Figure 7.6: Message Storage: Closing and Destroying Storage Connection

7.20 Message Storage: Creating and Storing A Message

Precondition: Storage connection is created and connected with messaging framework.

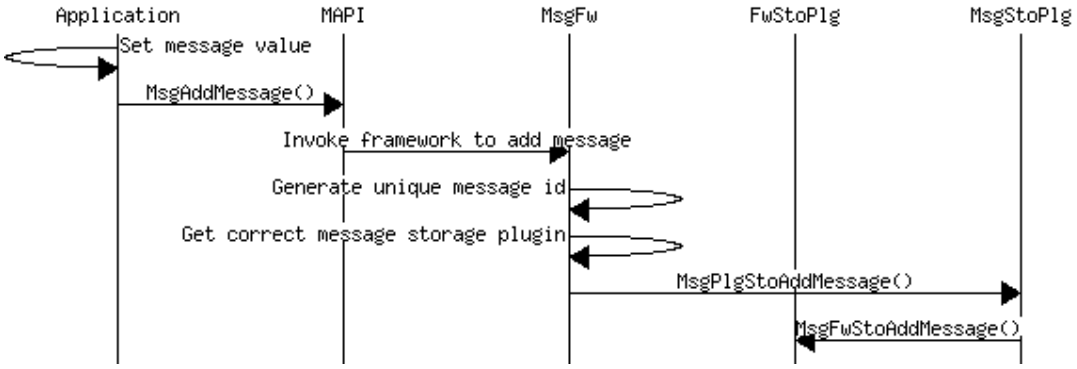

Figure 7.7: Message Storage: Creating and Storing a Message

7.21 Message Storage: Querying Messages

Precondition: Storage connection is created and connected with messaging framework

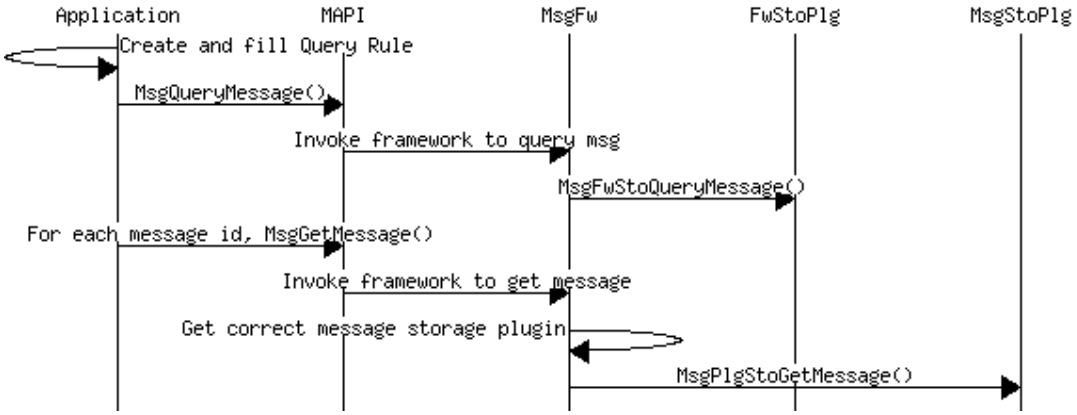

Figure 7.8: Message Storage: Querying Messages

7.22 Folder Storage: Creating and Storing a Folder

Precondition: Storage connection is created and connected with messaging framework

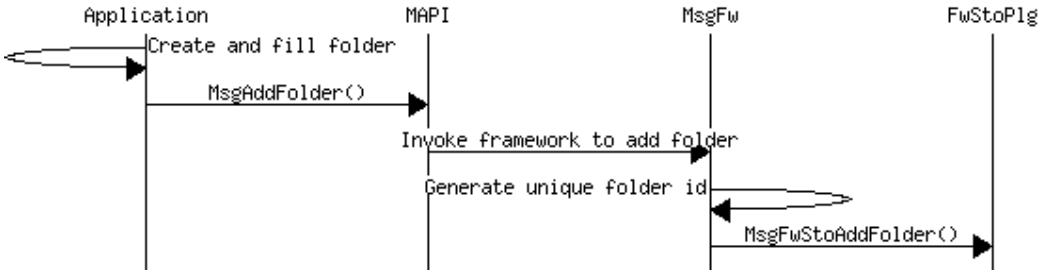

Figure 7.9: Messaging Framework Startup

7.23 Account Storage: Creating and Storing an Account

Precondition: Storage connection is created and connected with messaging framework

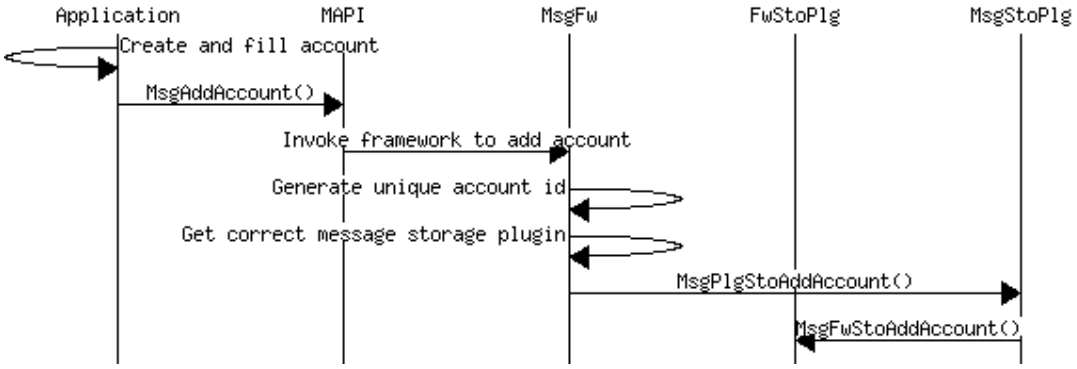

Figure 7.10: Account Storage: Creating and Storing an Account

7.24 Filter Plug-in: Set Filter List to Filter Messages

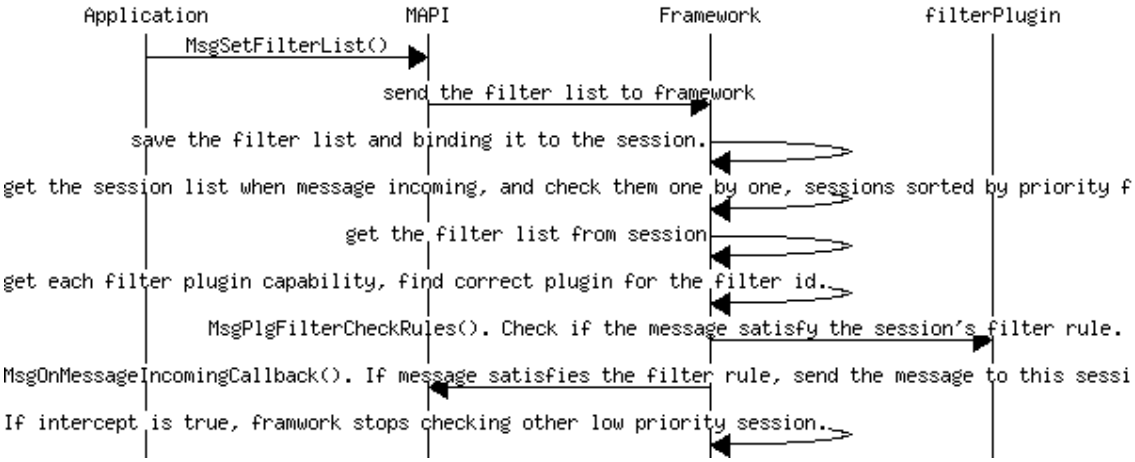

Figure 7.11: Filter Plug-in: Set Filter List to Filter Messages

7.25 Sending Messages (Normal Flow)

Precondition: Storage connection is created and connected with messaging framework

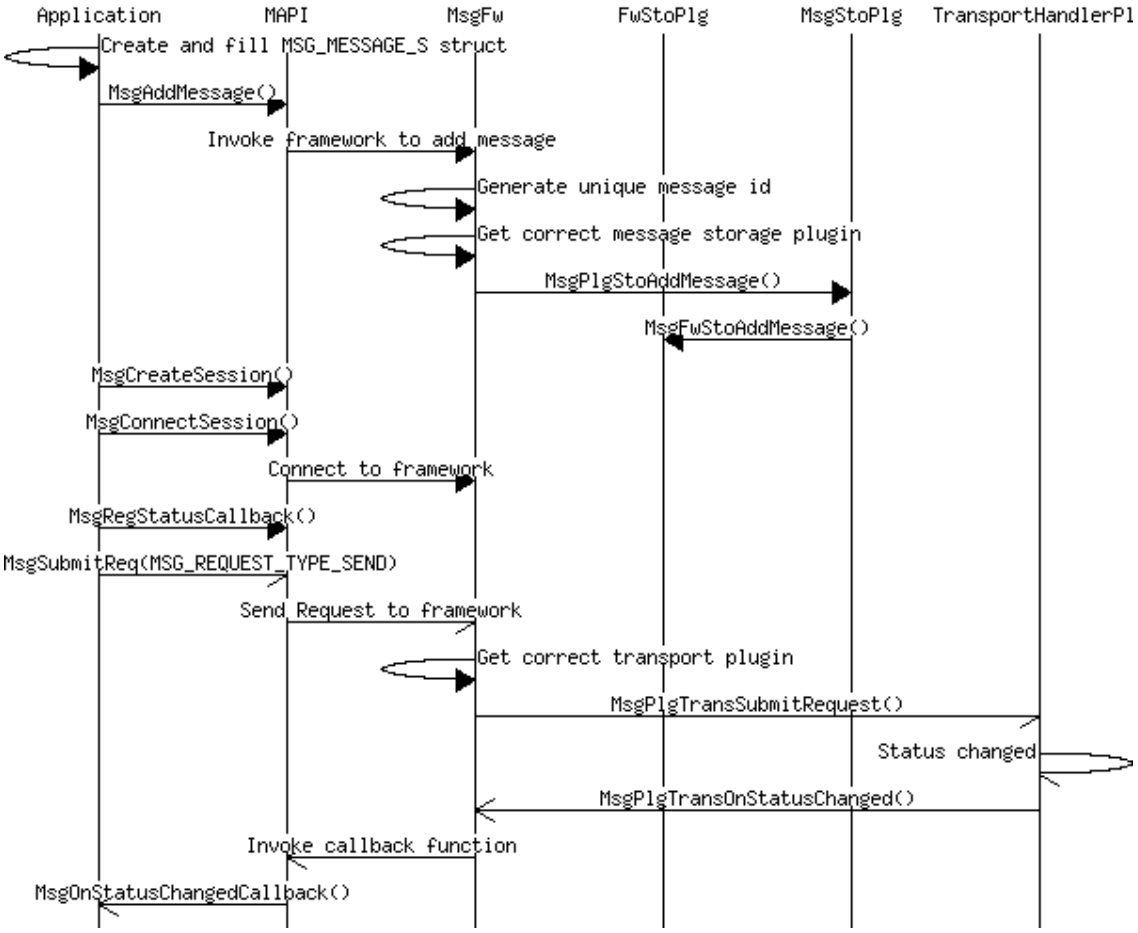

Figure 7.12: Sending Messages (Normal Flow)

7.26 Sending Messages (Alternative Flow)

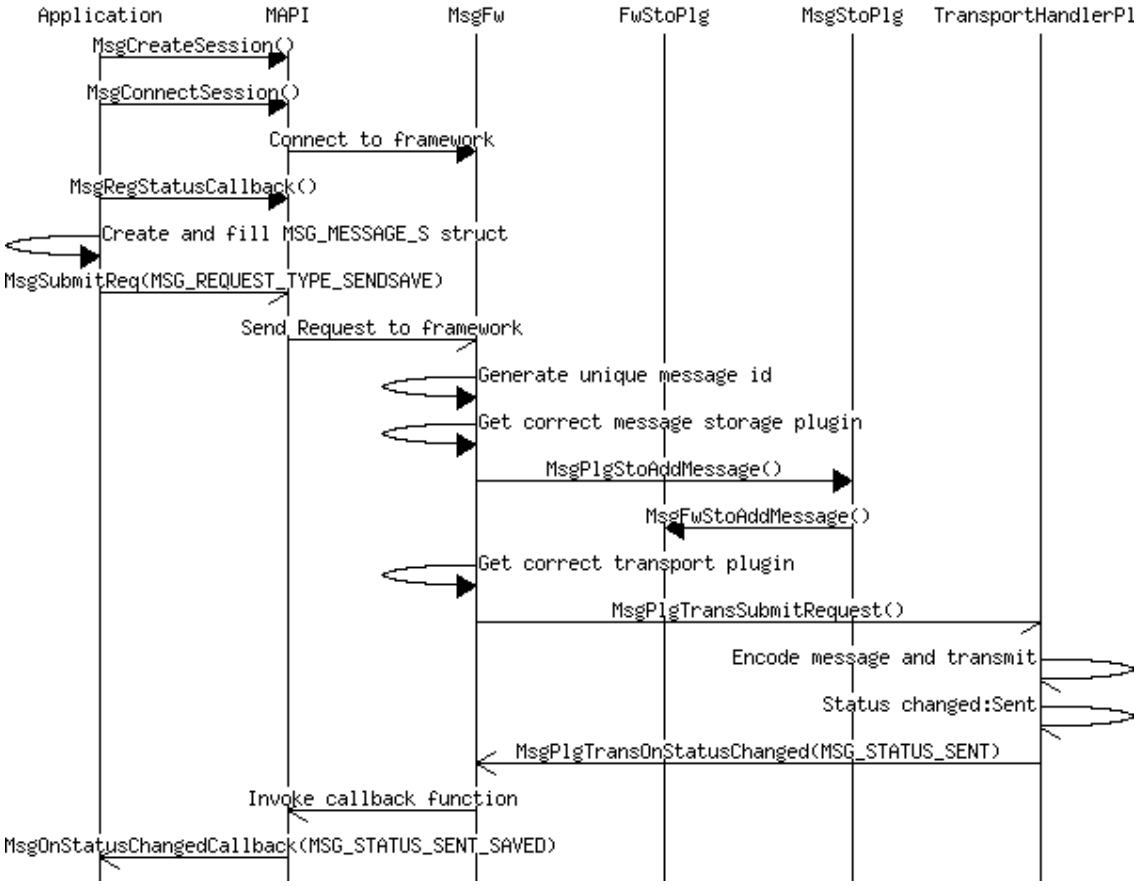

Figure 7.13: Sending Messages (Alternative Flow)

7.27 Receiving Messages

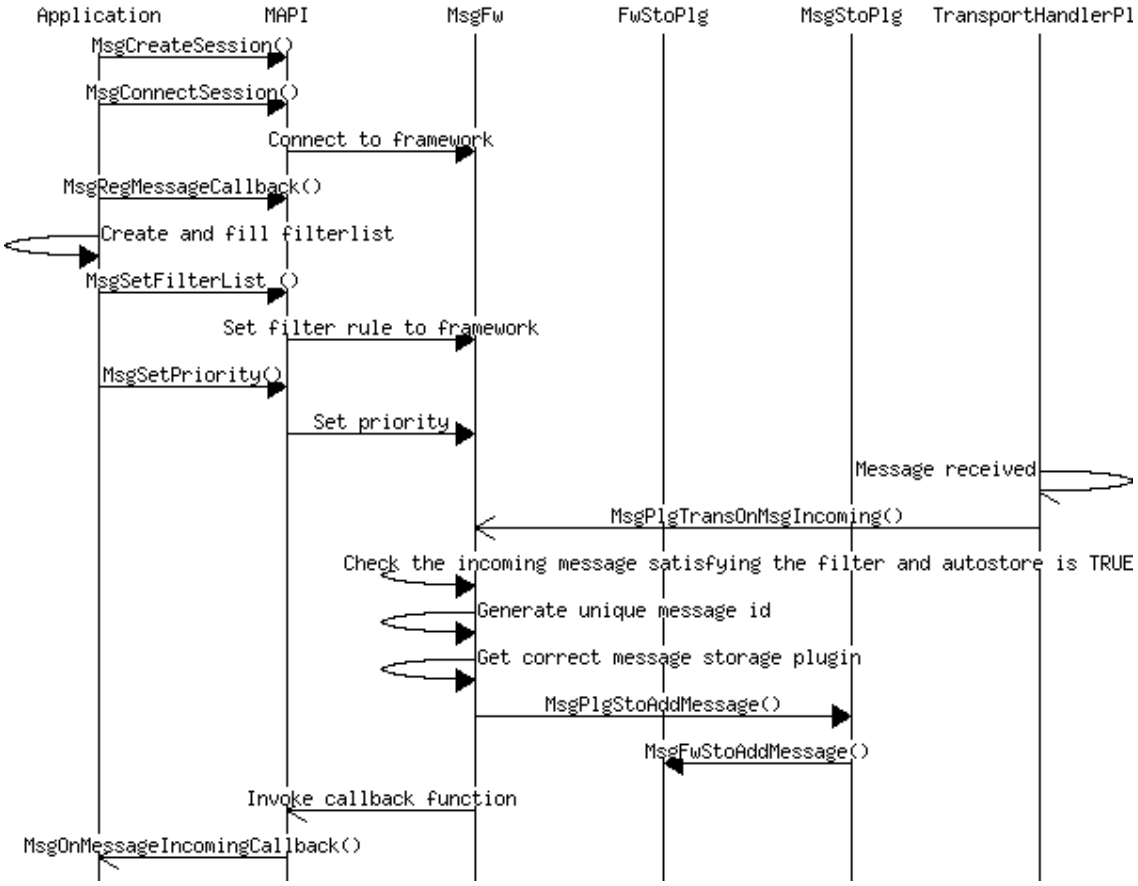

Figure 7.14: Receiving Messages

7.28 Explore Message List (Flow for Listing Messages)

Precondition: Storage connection is created and connected with messaging framework

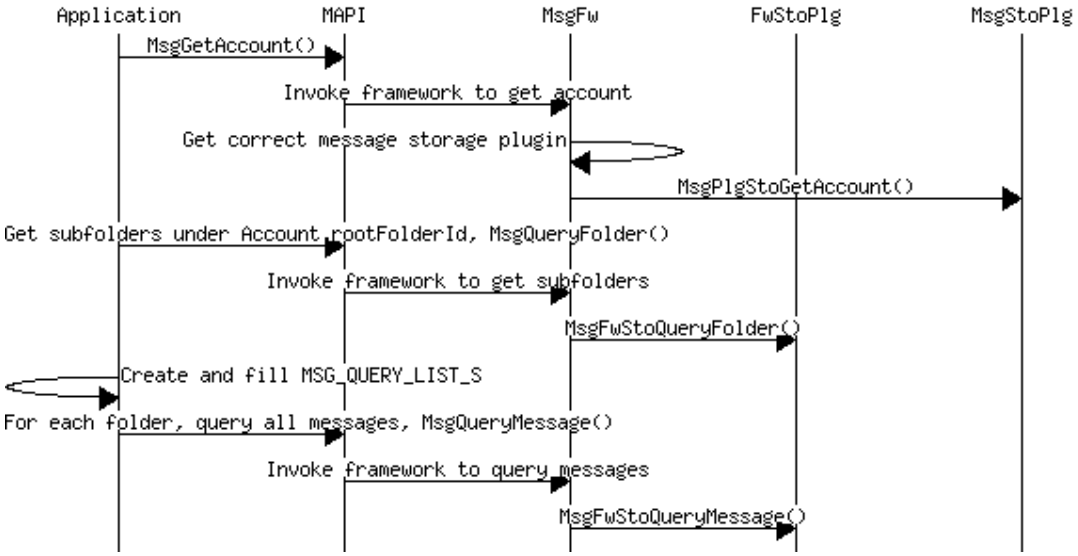

Figure 7.15: Explore Message List (Flow for Listing Messages)

7.29 Explore Message List (Flow for Browsing a Message)

Precondition: Storage connection is created and connected with messaging framework

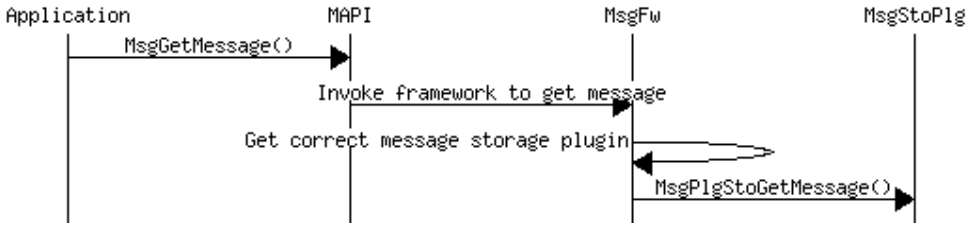

Figure 7.16: Explore Message List (Flow for Browsing a Message)

7.30 Explore Message List (Flow for Removal of a Message)

Precondition: Storage connection is created and connected with messaging framework

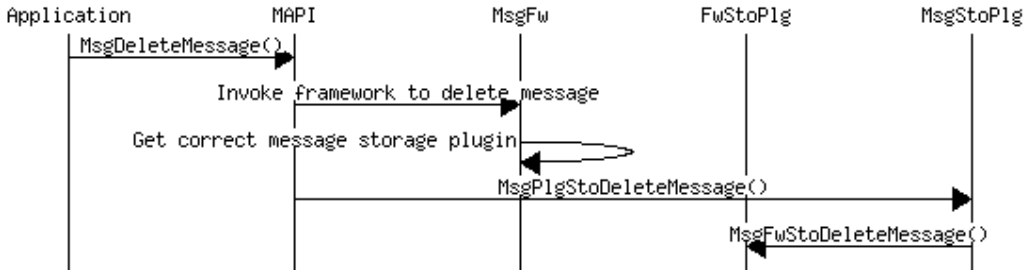

Figure 7.17: Explore Message List (Flow for Removal of a Message)

7.31 Explore Message List (Flow for Moving Messages Between Folders)

Precondition: Storage connection is created and connected with messaging framework

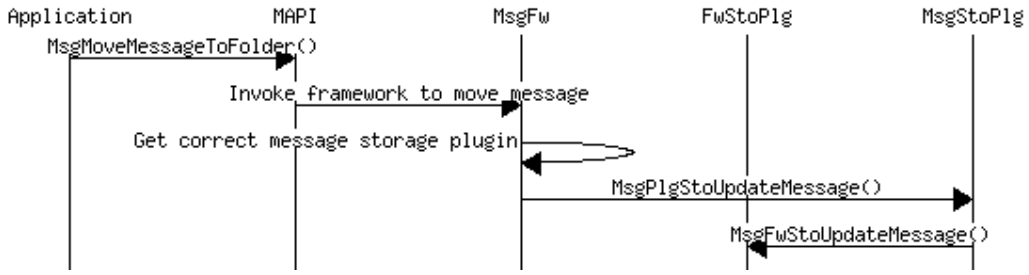

Figure 7.18: Explore Message List (Flow for Moving Messages Between Folders)

7.32 Manage Account and Profile (Flow for Creation of an Account)

Precondition: Storage connection is created and connected with messaging framework

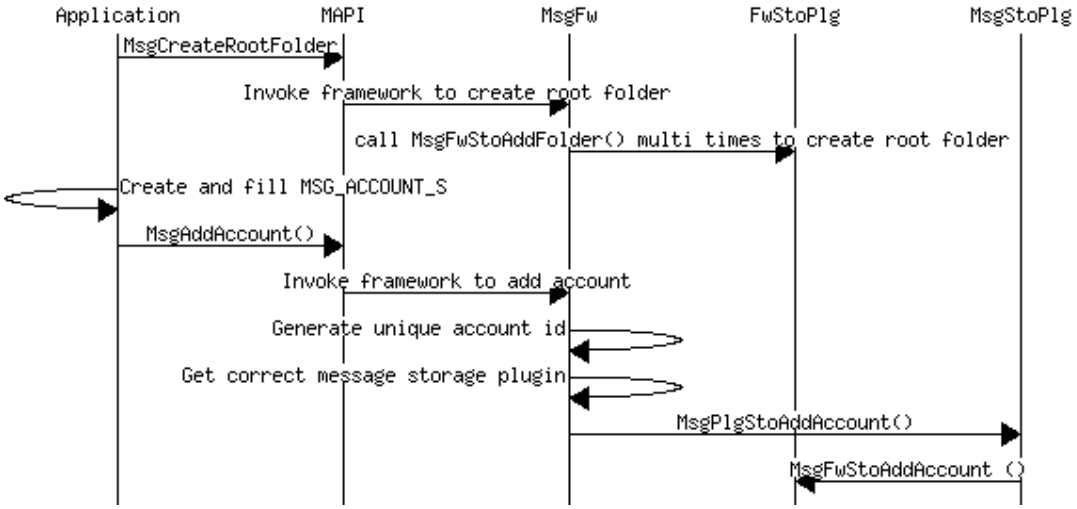

Figure 7.19: Manage Account and Profile (Flow for Creation of an Account)

7.33 Manage Account and Profile (Flow for Update of an Account)

Precondition: Storage connection is created and connected with messaging framework

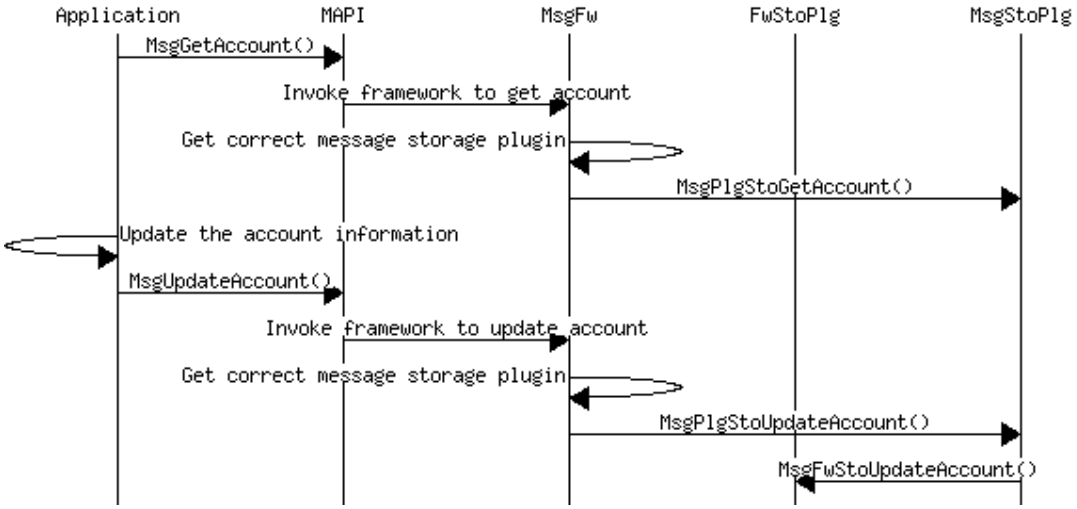

Figure 7.20: Manage Account and Profile (Flow for Update of an Account)

7.34 Manage Account and Profile (Setting Default Account)

Precondition: Storage connection is created and connected with messaging framework

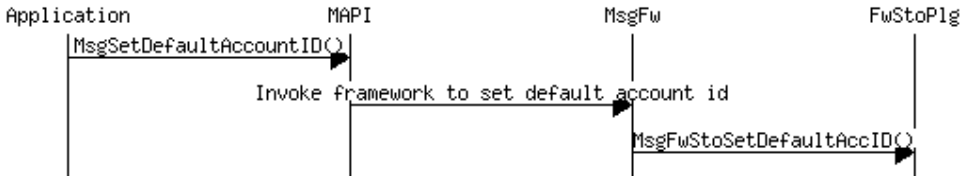

Figure 7.21: Manage Account and Profile (Setting Default Account)

7.35 Manage Account and Profile (Flow for Removing an Account)

Precondition: Storage connection is created and connected with messaging framework.

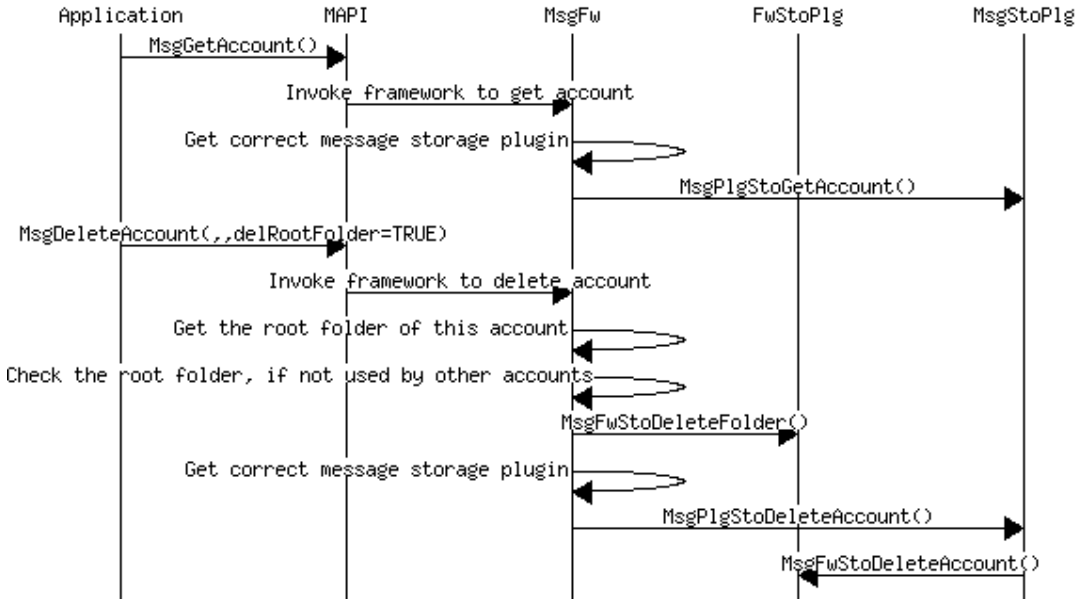

Figure 7.22: Manage Account and Profile (Flow for Removing an Account)

7.36 Manage Account and Profile (Alternative Flow for Removing an Account)

Precondition: Storage connection is created and connected with messaging framework

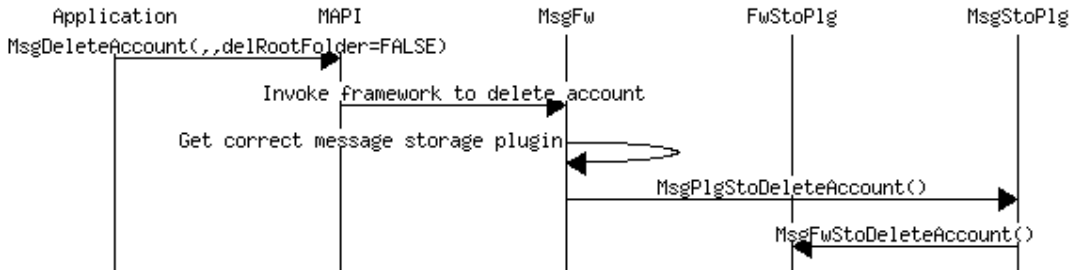

Figure 7.23: Manage Account and Profile (Alternative Flow for Removing an Account)

Index

_MSG_ACCOUNT_ID_LIST_S, 9
    accountIds, 9
    iCount, 9
_MSG_ACCOUNT_S, 10
    accountId, 10
    accountName, 10
    defaultMsgStorageId, 10
    extDataSize, 11
    msgType, 10
    pExtData, 11
    rootFolderId, 10
    storageId, 10
_MSG_CONTACT_LIST_S, 12
    iCount, 12
    pContacts, 12
_MSG_CONTACT_S, 13
    contactName, 13
    contactType, 13
    contactVal, 13
    extDataSize, 13
    pExtData, 13
_MSG_CONTACT_TYPE_E
    Msg_Mapi_Types.h, 126
_MSG_ENCODE_TYPE_E
    Msg_Mapi_Types.h, 127
_MSG_ERROR_E
    Msg_Mapi_Types.h, 127
_MSG_FILTER_ID_E
    Msg_Mapi_FilterRule.h, 45
_MSG_FILTER_ID_LIST_S, 15
    filterIds, 15
    iCount, 15
_MSG_FILTER_LIST_S, 16
    filters, 16
    iCount, 16
_MSG_FILTER_S, 17
    autoStore, 17
    filterId, 17
    filterName, 17
    filterValue, 18
    intercept, 17
    valueLength, 18
_MSG_FOLDER_ID_LIST_S, 19
    folderIds, 19
    iCount, 19
_MSG_FOLDER_S, 20
    commExtDataSize, 20
    folderId, 20
    folderName, 20
    folderType, 20
    parentId, 20
    pCommExtData, 21
    sortType, 20
    storageId, 20
_MSG_FOLDER_TYPE_E
    Msg_Mapi_Types.h, 128
_MSG_MAIN_TYPE_E
    Msg_Mapi_Types.h, 128
_MSG_MEDIA_TYPE_E
    Msg_Mapi_Types.h, 129
_MSG_MESSAGE_ID_LIST_S, 22
    iCount, 22
    msgIds, 22
_MSG_MESSAGE_PRIORITY_E
    Msg_Mapi_Types.h, 129
_MSG_MESSAGE_REQUEST_S, 23
    extDataSize, 23
    msgType, 23
    pExtData, 24
    pMsg, 23
    reqId, 23
    reqType, 23
_MSG_MESSAGE_S, 25
    accountId, 26
    commExtDataSize, 27
    deliveryStatus, 27
    displayFrom, 26
    displaySubject, 26
    displayTime, 26
    displayTo, 26
    extDataSize, 27
    from, 26
    hasAttachment, 27
    hasDrmContent, 27
    msgId, 25
    msgType, 25
    parentId, 25
    pCommExtData, 27
    pExtData, 27
    priority, 27

size, 27
status, 27
storageId, 26
subject, 26
to, 26
_MSG_MESSAGE_STATUS_S, 29
error, 29
extDataSize, 29
isFinal, 29
msgId, 29
pExtData, 30
reqId, 29
status, 29
_MSG_MESSAGE_TYPE_LIST_S, 31
iCount, 31
pMsgTypes, 31
_MSG_MESSAGE_TYPE_S, 32
mainType, 32
subType, 32
_MSG_QUERY_ID_E
Msg_Mapi_QueryRule.h, 53
_MSG_QUERY_ID_LIST_S, 33
iCount, 33
queryIds, 33
_MSG_QUERY_LIST_S, 34
iCount, 34
queries, 34
_MSG_QUERY_S, 35
queryId, 35
queryValue, 35
valueLength, 35
_MSG_REQUEST_TYPE_E
Msg_Mapi_Types.h, 129
_MSG_SESSION_PRIORITY_E
Msg_Mapi_Types.h, 130
_MSG_SIZE_UNIT_E
Msg_Mapi_Types.h, 130
_MSG_SORT_RULE_S, 36
isAscending, 36
sortType, 36
_MSG_SORT_TYPE_E
Msg_Mapi_QueryRule.h, 53
_MSG_SORT_TYPE_LIST_S, 37
iCount, 37
sortTypes, 37
_MSG_STATUS_E
Msg_Mapi_Types.h, 130
_MSG_STORAGE_ID_LIST_S, 38
iCount, 38
pStorageIds, 38
_MSG_STORAGE_ID_VALUE_E
Msg_Mapi_Types.h, 131
_MSG_STORAGE_REGISTRY_S, 39
canDeregister, 39
media_type, 39
name, 39
path, 39
storeId, 39
_MSG_STRING_S, 40
encodeType, 40
length, 40
pString, 40

accountId
_MSG_ACCOUNT_S, 10
_MSG_MESSAGE_S, 26
accountIds
_MSG_ACCOUNT_ID_LIST_S, 9
accountName
_MSG_ACCOUNT_S, 10
autoStore
_MSG_FILTER_S, 17

bool
Msg_Mapi_Types.h, 123

canDeregister
_MSG_STORAGE_REGISTRY_S, 39
commExtDataSize
_MSG_FOLDER_S, 20
_MSG_MESSAGE_S, 27
COMPONENT_OVERVIEW_H
ComponentOverview.h, 41
ComponentOverview.h, 41
COMPONENT_OVERVIEW_H, 41
contactName
_MSG_CONTACT_S, 13
contactType
_MSG_CONTACT_S, 13
contactVal
_MSG_CONTACT_S, 13

defaultMsgStorageId
_MSG_ACCOUNT_S, 10
deliveryStatus
_MSG_MESSAGE_S, 27
displayFrom
_MSG_MESSAGE_S, 26
displaySubject
_MSG_MESSAGE_S, 26
displayTime
_MSG_MESSAGE_S, 26
displayTo
_MSG_MESSAGE_S, 26

encodeType
_MSG_STRING_S, 40
error
_MSG_MESSAGE_STATUS_S, 29

extDataSize
_MSG_ACCOUNT_S, 11
_MSG_CONTACT_S, 13
_MSG_MESSAGE_REQUEST_S, 23
_MSG_MESSAGE_S, 27
_MSG_MESSAGE_STATUS_S, 29

false
Msg_Mapi_Types.h, 123
filterId
_MSG_FILTER_S, 17
filterIds
_MSG_FILTER_ID_LIST_S, 15
filterName
_MSG_FILTER_S, 17
filters
_MSG_FILTER_LIST_S, 16
filterValue
_MSG_FILTER_S, 18
folderId
_MSG_FOLDER_S, 20
folderIds
_MSG_FOLDER_ID_LIST_S, 19
folderName
_MSG_FOLDER_S, 20
folderType
_MSG_FOLDER_S, 20
from
_MSG_MESSAGE_S, 26

hasAttachment
_MSG_MESSAGE_S, 27
hasDrmContent
_MSG_MESSAGE_S, 27

iCount
_MSG_ACCOUNT_ID_LIST_S, 9
_MSG_CONTACT_LIST_S, 12
_MSG_FILTER_ID_LIST_S, 15
_MSG_FILTER_LIST_S, 16
_MSG_FOLDER_ID_LIST_S, 19
_MSG_MESSAGE_ID_LIST_S, 22
_MSG_MESSAGE_TYPE_LIST_S, 31
_MSG_QUERY_ID_LIST_S, 33
_MSG_QUERY_LIST_S, 34
_MSG_SORT_TYPE_LIST_S, 37
_MSG_STORAGE_ID_LIST_S, 38
intercept
_MSG_FILTER_S, 17
isAscending
_MSG_SORT_RULE_S, 36
isFinal
_MSG_MESSAGE_STATUS_S, 29

length
_MSG_STRING_S, 40

mainType
_MSG_MESSAGE_TYPE_S, 32
media_type
_MSG_STORAGE_REGISTRY_S, 39
MSG_CONTACT_TYPE_EMAIL
Msg_Mapi_Types.h, 126
MSG_CONTACT_TYPE_PLMN
Msg_Mapi_Types.h, 127
MSG_CONTACT_TYPE_UNKNOWN
Msg_Mapi_Types.h, 126
MSG_ENCODE_ASCII
Msg_Mapi_Types.h, 127
MSG_ENCODE_GSM7BIT
Msg_Mapi_Types.h, 127
MSG_ENCODE_SHIFT_JIS
Msg_Mapi_Types.h, 127
MSG_ENCODE_UCS2
Msg_Mapi_Types.h, 127
MSG_ENCODE_UTF16
Msg_Mapi_Types.h, 127
MSG_ENCODE_UTF8
Msg_Mapi_Types.h, 127
MSG_ERR_COMMUNICATION_ERROR
Msg_Mapi_Types.h, 128
MSG_ERR_CONNECTION_NOT_CONNECT
Msg_Mapi_Types.h, 128
MSG_ERR_FRAMEWORK_RESERVE_BEGIN
Msg_Mapi_Types.h, 128
MSG_ERR_FRAMEWORK_RESERVE_END
Msg_Mapi_Types.h, 128
MSG_ERR_FS_ERROR
Msg_Mapi_Types.h, 127
MSG_ERR_FS_FULL
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_ACCOUNT
Msg_Mapi_Types.h, 128
MSG_ERR_INVALID_ACCOUNT_ID
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_CONNECTION
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_FILTER_ID
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_FOLDER
Msg_Mapi_Types.h, 128
MSG_ERR_INVALID_FOLDER_ID
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_FOLDER_TYPE
Msg_Mapi_Types.h, 128
MSG_ERR_INVALID_MESSAGE
Msg_Mapi_Types.h, 128
MSG_ERR_INVALID_MESSAGE_ID
Msg_Mapi_Types.h, 127

MSG_ERR_INVALID_MSG_TYPE
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_QUERY_ID
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_REQUEST
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_REQUEST_ID
Msg_Mapi_Types.h, 128
MSG_ERR_INVALID_SESSION_HANDLE
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_SESSION_PRIORITY
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_SORT_TYPE
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_STORAGE
Msg_Mapi_Types.h, 127
MSG_ERR_INVALID_STORAGE_REG
Msg_Mapi_Types.h, 127
MSG_ERR_MAX_NUMBER_REACHED
Msg_Mapi_Types.h, 128
MSG_ERR_MEMORY_ERROR
Msg_Mapi_Types.h, 128
MSG_ERR_NOT_EMPTY_FOLDER
Msg_Mapi_Types.h, 128
MSG_ERR_NULL_POINTER
Msg_Mapi_Types.h, 127
MSG_ERR_PERMISSION_DENIED
Msg_Mapi_Types.h, 127
MSG_ERR_PLUGIN
Msg_Mapi_Types.h, 128
MSG_ERR_PLUGIN_DEF_BEGIN_VALUE
Msg_Mapi_Types.h, 128
MSG_ERR_ROOTFOLDER_INUSE
Msg_Mapi_Types.h, 128
MSG_ERR_SESSION_NOT_CONNECTED
Msg_Mapi_Types.h, 127
MSG_ERR_STORAGE_ERROR
Msg_Mapi_Types.h, 128
MSG_ERR_STORAGE_FULL
Msg_Mapi_Types.h, 128
MSG_ERR_STORAGE_NOT_INIT
Msg_Mapi_Types.h, 127
MSG_ERR_STORAGE_REG_NOT_INIT
Msg_Mapi_Types.h, 127
MSG_ERR_UNKNOWN
Msg_Mapi_Types.h, 128
MSG_FILTER_BY_FROM
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_BY_HAS_ATTACHMENT
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_BY_HAS_DRM_CONTENT
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_BY_MSG_TYPE
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_BY_PRIORITY_GT
Msg_Mapi_FilterRule.h, 46
MSG_FILTER_BY_PRIORITY_LT
Msg_Mapi_FilterRule.h, 46
MSG_FILTER_BY_SIZE_GT
Msg_Mapi_FilterRule.h, 46
MSG_FILTER_BY_SIZE_LT
Msg_Mapi_FilterRule.h, 46
MSG_FILTER_BY_UNKNOWN
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_BY_USER_DEF_BEGIN
Msg_Mapi_FilterRule.h, 46
MSG_FOLDER_TYPE_DELETED
Msg_Mapi_Types.h, 128
MSG_FOLDER_TYPE_DRAFTS
Msg_Mapi_Types.h, 128
MSG_FOLDER_TYPE_INBOX
Msg_Mapi_Types.h, 128
MSG_FOLDER_TYPE_OUTBOX
Msg_Mapi_Types.h, 128
MSG_FOLDER_TYPE_ROOT
Msg_Mapi_Types.h, 128
MSG_FOLDER_TYPE_SENT
Msg_Mapi_Types.h, 128
MSG_FOLDER_TYPE_USERDEFINE
Msg_Mapi_Types.h, 128
Msg_Mapi_FilterRule.h
MSG_FILTER_BY_FROM, 45
MSG_FILTER_BY_HAS_ATTACHMENT, 45
MSG_FILTER_BY_HAS_DRM_CONTENT, 45
MSG_FILTER_BY_MSG_TYPE, 45
MSG_FILTER_BY_PRIORITY_GT, 46
MSG_FILTER_BY_PRIORITY_LT, 46
MSG_FILTER_BY_SIZE_GT, 46
MSG_FILTER_BY_SIZE_LT, 46
MSG_FILTER_BY_UNKNOWN, 45
MSG_FILTER_BY_USER_DEF_BEGIN, 46
Msg_Mapi_QueryRule.h
MSG_QUERY_BY_AFTER_DATETIME, 53
MSG_QUERY_BY_BEFORE_DATETIME, 53
MSG_QUERY_BY_DELIVERY_STATUS, 53
MSG_QUERY_BY_FROM, 53
MSG_QUERY_BY_HASATTACHMENT, 53
MSG_QUERY_BY_HASDRMCONTENT, 53
MSG_QUERY_BY_PRIORITY, 53
MSG_QUERY_BY_SIZE, 53
MSG_QUERY_BY_STATUS, 53
MSG_QUERY_BY_SUBJECT, 53
MSG_QUERY_BY_TO, 53

MSG_QUERY_BY_UNKNOWN, 53
MSG_QUERY_BY_USER_DEF_BEGIN, 53
MSG_SORT_BY_DELIVERY_STATUS, 53
MSG_SORT_BY_DISPLAY_FROM, 53
MSG_SORT_BY_DISPLAY_SUBJECT, 53
MSG_SORT_BY_DISPLAY_TIME, 53
MSG_SORT_BY_DISPLAY_TO, 53
MSG_SORT_BY_HASATTACHMENT, 53
MSG_SORT_BY_HASDRMCONTENT, 53
MSG_SORT_BY_PRIORITY, 53
MSG_SORT_BY_SIZE, 53
MSG_SORT_BY_STATUS, 53
MSG_SORT_BY_UNKNOWN, 53
MSG_SORT_BY_USER_DEF_BEGIN, 53
Msg_Mapi_Types.h
MSG_CONTACT_TYPE_EMAIL, 126
MSG_CONTACT_TYPE_PLMN, 127
MSG_CONTACT_TYPE_UNKNOWN, 126
MSG_ENCODE_ASCII, 127
MSG_ENCODE_GSM7BIT, 127
MSG_ENCODE_SHIFT_JIS, 127
MSG_ENCODE_UCS2, 127
MSG_ENCODE_UTF16, 127
MSG_ENCODE_UTF8, 127
MSG_ERR_COMMUNICATION_ERROR, 128
MSG_ERR_CONNECTION_NOT_-CONNECT, 128
MSG_ERR_FRAMEWORK_RESERVE_-BEGIN, 128
MSG_ERR_FRAMEWORK_RESERVE_-END, 128
MSG_ERR_FS_ERROR, 127
MSG_ERR_FS_FULL, 127
MSG_ERR_INVALID_ACCOUNT, 128
MSG_ERR_INVALID_ACCOUNT_ID, 127
MSG_ERR_INVALID_CONNECTION, 127
MSG_ERR_INVALID_FILTER_ID, 127
MSG_ERR_INVALID_FOLDER, 128
MSG_ERR_INVALID_FOLDER_ID, 127
MSG_ERR_INVALID_FOLDER_TYPE, 128
MSG_ERR_INVALID_MESSAGE, 128
MSG_ERR_INVALID_MESSAGE_ID, 127
MSG_ERR_INVALID_MSG_TYPE, 127
MSG_ERR_INVALID_QUERY_ID, 127
MSG_ERR_INVALID_REQUEST, 127
MSG_ERR_INVALID_REQUEST_ID, 128
MSG_ERR_INVALID_SESSION_HANDLE, 127
MSG_ERR_INVALID_SESSION_-PRIORITY, 127
MSG_ERR_INVALID_SORT_TYPE, 127
MSG_ERR_INVALID_STORAGE, 127
MSG_ERR_INVALID_STORAGE_REG, 127
MSG_ERR_MAX_NUMBER_REACHED, 128
MSG_ERR_MEMORY_ERROR, 128
MSG_ERR_NOT_EMPTY_FOLDER, 128
MSG_ERR_NULL_POINTER, 127
MSG_ERR_PERMISSION_DENIED, 127
MSG_ERR_PLUGIN, 128
MSG_ERR_PLUGIN_DEF_BEGIN_VALUE, 128
MSG_ERR_ROOTFOLDER_INUSE, 128
MSG_ERR_SESSION_NOT_CONNECTED, 127
MSG_ERR_STORAGE_ERROR, 128
MSG_ERR_STORAGE_FULL, 128
MSG_ERR_STORAGE_NOT_INIT, 127
MSG_ERR_STORAGE_REG_NOT_INIT, 127
MSG_ERR_UNKNOWN, 128
MSG_FOLDER_TYPE_DELETED, 128
MSG_FOLDER_TYPE_DRAFTS, 128
MSG_FOLDER_TYPE_INBOX, 128
MSG_FOLDER_TYPE_OUTBOX, 128
MSG_FOLDER_TYPE_ROOT, 128
MSG_FOLDER_TYPE_SENT, 128
MSG_FOLDER_TYPE_USERDEFINE, 128
MSG_MEDIA_FLASH, 129
MSG_MEDIA_REMOVABLE, 129
MSG_MEDIA_SIM, 129
MSG_MESSAGE_PRIORITY_HIGH, 129
MSG_MESSAGE_PRIORITY_LOW, 129
MSG_MESSAGE_PRIORITY_NORMAL, 129
MSG_MESSAGE_TYPE_EMAIL, 129
MSG_MESSAGE_TYPE_IMODE, 129
MSG_MESSAGE_TYPE_MMS, 129
MSG_MESSAGE_TYPE_SMS, 129
MSG_MESSAGE_TYPE_UNKNOWN, 129
MSG_REQUEST_TYPE_INVALID, 129
MSG_REQUEST_TYPE_RESERVED_-BEGIN, 129
MSG_REQUEST_TYPE_RESERVED_END, 130
MSG_REQUEST_TYPE_SEND, 129
MSG_REQUEST_TYPE_SENDSAVE, 129
MSG_REQUEST_TYPE_USER_DEF_-BEGIN, 130
MSG_SESSION_PRIORITY_HIGH, 130
MSG_SESSION_PRIORITY_HIGHEST, 130
MSG_SESSION_PRIORITY_LOW, 130
MSG_SESSION_PRIORITY_LOWEST, 130
MSG_SESSION_PRIORITY_NORMAL, 130
MSG_SIZE_UNIT_BYTE, 130
MSG_SIZE_UNIT_GBYTE, 130
MSG_SIZE_UNIT_KBYTE, 130

MSG_SIZE_UNIT_MBYTE, 130
MSG_SIZE_UNIT_NUMBER, 130
MSG_STATUS_DRAFT, 131
MSG_STATUS_EDITING, 131
MSG_STATUS_FAIL_TO_SEND, 131
MSG_STATUS_FAIL_TO_SENDSAVE, 131
MSG_STATUS_FORWARDED, 131
MSG_STATUS_PROTECTED, 131
MSG_STATUS_READ, 130
MSG_STATUS_REPLIED, 131
MSG_STATUS_REQ_CANCELLED, 131
MSG_STATUS_SAVED_NOTSENT, 131
MSG_STATUS_SENDING, 131
MSG_STATUS_SENT, 131
MSG_STATUS_SENT_NOTSAVED, 131
MSG_STATUS_SENT_SAVED, 131
MSG_STATUS_UNKNOWN, 130
MSG_STATUS_UNREAD, 131
MSG_STATUS_USER_DEF_BEGIN, 131
MSG_STORAGE_DEFAULT, 131
MSG_STORAGE_UNKNOWN, 131
MSG_SUCCESS, 127
MSG_MEDIA_FLASH
Msg_Mapi_Types.h, 129
MSG_MEDIA_REMOVABLE
Msg_Mapi_Types.h, 129
MSG_MEDIA_SIM
Msg_Mapi_Types.h, 129
MSG_MESSAGE_PRIORITY_HIGH
Msg_Mapi_Types.h, 129
MSG_MESSAGE_PRIORITY_LOW
Msg_Mapi_Types.h, 129
MSG_MESSAGE_PRIORITY_NORMAL
Msg_Mapi_Types.h, 129
MSG_MESSAGE_TYPE_EMAIL
Msg_Mapi_Types.h, 129
MSG_MESSAGE_TYPE_IMODE
Msg_Mapi_Types.h, 129
MSG_MESSAGE_TYPE_MMS
Msg_Mapi_Types.h, 129
MSG_MESSAGE_TYPE_SMS
Msg_Mapi_Types.h, 129
MSG_MESSAGE_TYPE_UNKNOWN
Msg_Mapi_Types.h, 129
MSG_QUERY_BY_AFTER_DATETIME
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_BEFORE_DATETIME
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_DELIVERY_STATUS
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_FROM
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_HASATTACHMENT
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_HASDRMCONTENT
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_PRIORITY
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_SIZE
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_STATUS
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_SUBJECT
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_TO
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_UNKNOWN
Msg_Mapi_QueryRule.h, 53
MSG_QUERY_BY_USER_DEF_BEGIN
Msg_Mapi_QueryRule.h, 53
MSG_REQUEST_TYPE_INVALID
Msg_Mapi_Types.h, 129
MSG_REQUEST_TYPE_RESERVED_BEGIN
Msg_Mapi_Types.h, 129
MSG_REQUEST_TYPE_RESERVED_END
Msg_Mapi_Types.h, 130
MSG_REQUEST_TYPE_SEND
Msg_Mapi_Types.h, 129
MSG_REQUEST_TYPE_SENDSAVE
Msg_Mapi_Types.h, 129
MSG_REQUEST_TYPE_USER_DEF_BEGIN
Msg_Mapi_Types.h, 130
MSG_SESSION_PRIORITY_HIGH
Msg_Mapi_Types.h, 130
MSG_SESSION_PRIORITY_HIGHEST
Msg_Mapi_Types.h, 130
MSG_SESSION_PRIORITY_LOW
Msg_Mapi_Types.h, 130
MSG_SESSION_PRIORITY_LOWEST
Msg_Mapi_Types.h, 130
MSG_SESSION_PRIORITY_NORMAL
Msg_Mapi_Types.h, 130
MSG_SIZE_UNIT_BYTE
Msg_Mapi_Types.h, 130
MSG_SIZE_UNIT_GBYTE
Msg_Mapi_Types.h, 130
MSG_SIZE_UNIT_KBYTE
Msg_Mapi_Types.h, 130
MSG_SIZE_UNIT_MBYTE
Msg_Mapi_Types.h, 130
MSG_SIZE_UNIT_NUMBER
Msg_Mapi_Types.h, 130
MSG_SORT_BY_DELIVERY_STATUS
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_DISPLAY_FROM
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_DISPLAY_SUBJECT
Msg_Mapi_QueryRule.h, 53

MSG_SORT_BY_DISPLAY_TIME
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_DISPLAY_TO
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_HASATTACHMENT
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_HASDRMCONTENT
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_PRIORITY
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_SIZE
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_STATUS
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_UNKNOWN
Msg_Mapi_QueryRule.h, 53
MSG_SORT_BY_USER_DEF_BEGIN
Msg_Mapi_QueryRule.h, 53
MSG_STATUS_DRAFT
Msg_Mapi_Types.h, 131
MSG_STATUS_EDITING
Msg_Mapi_Types.h, 131
MSG_STATUS_FAIL_TO_SEND
Msg_Mapi_Types.h, 131
MSG_STATUS_FAIL_TO_SENDSAVE
Msg_Mapi_Types.h, 131
MSG_STATUS_FORWARDED
Msg_Mapi_Types.h, 131
MSG_STATUS_PROTECTED
Msg_Mapi_Types.h, 131
MSG_STATUS_READ
Msg_Mapi_Types.h, 130
MSG_STATUS_REPLIED
Msg_Mapi_Types.h, 131
MSG_STATUS_REQ_CANCELLED
Msg_Mapi_Types.h, 131
MSG_STATUS_SAVED_NOTSENT
Msg_Mapi_Types.h, 131
MSG_STATUS_SENDING
Msg_Mapi_Types.h, 131
MSG_STATUS_SENT
Msg_Mapi_Types.h, 131
MSG_STATUS_SENT_NOTSAVED
Msg_Mapi_Types.h, 131
MSG_STATUS_SENT_SAVED
Msg_Mapi_Types.h, 131
MSG_STATUS_UNKNOWN
Msg_Mapi_Types.h, 130
MSG_STATUS_UNREAD
Msg_Mapi_Types.h, 131
MSG_STATUS_USER_DEF_BEGIN
Msg_Mapi_Types.h, 131
MSG_STORAGE_DEFAULT
Msg_Mapi_Types.h, 131
MSG_STORAGE_UNKNOWN
Msg_Mapi_Types.h, 131
MSG_SUCCESS
Msg_Mapi_Types.h, 127
MSG_ACCOUNT_ID_LIST_S
Msg_Mapi_Types.h, 124
MSG_ACCOUNT_ID_T
Msg_Mapi_Types.h, 124
MSG_ACCOUNT_S
Msg_Mapi_Account.h, 42
MSG_CONNECTION_HANDLE_T
Msg_Mapi_Storage_Manager.h, 93
MSG_CONTACT_LIST_S
Msg_Mapi_Contact.h, 43
MSG_CONTACT_S
Msg_Mapi_Contact.h, 43
MSG_CONTACT_TYPE_T
Msg_Mapi_Types.h, 124
MSG_DEFAULT_ACCOUNT_ID
Msg_Mapi_Types.h, 123
MSG_DEFAULT_FOLDER_ID
Msg_Mapi_Types.h, 123
MSG_ENCODE_TYPE_T
Msg_Mapi_Types.h, 124
MSG_ERROR_BEGIN
Msg_Mapi_Types.h, 123
MSG_ERROR_T
Msg_Mapi_Types.h, 124
MSG_FILTER_ID_LIST_S
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_ID_T
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_LIST_S
Msg_Mapi_FilterRule.h, 45
MSG_FILTER_S
Msg_Mapi_FilterRule.h, 45
MSG_FOLDER_ID_LIST_S
Msg_Mapi_Types.h, 124
MSG_FOLDER_ID_T
Msg_Mapi_Types.h, 125
MSG_FOLDER_S
Msg_Mapi_Folder.h, 47
MSG_FOLDER_TYPE_T
Msg_Mapi_Types.h, 125
MSG_INVALID_ACCOUNT_ID
Msg_Mapi_Types.h, 124
MSG_INVALID_FOLDER_ID
Msg_Mapi_Types.h, 124
MSG_INVALID_MESSAGE_ID
Msg_Mapi_Types.h, 124
MSG_MAIN_TYPE_T
Msg_Mapi_Types.h, 125
Msg_Mapi_Account.h, 42
MSG_ACCOUNT_S, 42

Msg_Mapi_Contact.h, 43
MSG_CONTACT_LIST_S, 43
MSG_CONTACT_S, 43
Msg_Mapi_FilterRule.h, 44
_MSG_FILTER_ID_E, 45
MSG_FILTER_ID_LIST_S, 45
MSG_FILTER_ID_T, 45
MSG_FILTER_LIST_S, 45
MSG_FILTER_S, 45
Msg_Mapi_Folder.h, 47
MSG_FOLDER_S, 47
Msg_Mapi_Message.h, 48
MSG_MESSAGE_S, 48
Msg_Mapi_Message_Request.h, 49
MSG_MESSAGE_REQUEST_S, 49
Msg_Mapi_Message_Status.h, 50
MSG_MESSAGE_STATUS_S, 50
Msg_Mapi_QueryRule.h, 51
_MSG_QUERY_ID_E, 53
_MSG_SORT_TYPE_E, 53
MSG_QUERY_ID_LIST_S, 52
MSG_QUERY_ID_T, 52
MSG_QUERY_LIST_S, 52
MSG_QUERY_S, 52
MSG_SORT_RULE_S, 52
MSG_SORT_TYPE_LIST_S, 52
MSG_SORT_TYPE_T, 52
Msg_Mapi_Session.h, 54
MSG_SESSION_HANDLE_T, 55
MsgCancelReq, 56
MsgConnectSession, 57
MsgCreateSession, 58
MsgDestroySession, 58
MsgDisconnectSession, 59
MsgGetAllSupportedFilterId, 59
MsgGetAllSupportedMsgType, 59
MsgGetSessionPluginError, 60
MsgOnMessageIncomingCallback, 55
MsgOnStatusChangedCallback, 56
MsgRegMessageCallback, 60
MsgRegStatusCallback, 61
MsgReleaseFilterIdList, 61
MsgReleaseMsgTypeList, 62
MsgSetFilterList, 62
MsgSetPriority, 63
MsgSubmitReq, 63
Msg_Mapi_Storage_Account.h, 66
MsgAddAccount, 67
MsgCountAccount, 68
MsgDeleteAccount, 68
MsgGetAccount, 69
MsgGetAccountIdsByStorage, 70
MsgGetCommAccount, 71
MsgGetDefaultAccountID, 71
MsgMoveAccountToStorage, 72
MsgQueryAccount, 73
MsgReleaseAccount, 74
MsgReleaseAccountIdList, 74
MsgReleaseCommAccount, 74
MsgSetDefaultAccountID, 74
MsgUpdateAccount, 75
Msg_Mapi_Storage_Folder.h, 77
MsgAddFolder, 78
MsgCountSubFolder, 79
MsgCountSubFolderRecursive, 80
MsgCreateRootFolder, 80
MsgDeleteFolder, 81
MsgGetAllRootFolderIds, 82
MsgGetFolder, 83
MsgGetFolderByType, 84
MsgGetFolderIdsByStorage, 84
MsgGetFolderParentId, 85
MsgGetPredefinedFolders, 86
MsgMoveFolderToFolder, 87
MsgMoveFolderToStorage, 87
MsgQueryFolder, 88
MsgQueryFolderRecursive, 89
MsgReleaseFolder, 90
MsgReleaseFolderIdList, 90
MsgUpdateFolder, 90
Msg_Mapi_Storage_Manager.h, 92
MSG_CONNECTION_HANDLE_T, 93
MsgCloseStoConnection, 93
MsgCreateStoConnection, 94
MsgDestroyStoConnection, 94
MsgGetFwStorageStatus, 95
MsgGetMediaTypeByStorageId, 96
MsgGetMsgStorageStatus, 96
MsgGetStoragePluginError, 97
MsgGetStorageRegistry, 97
MsgInitStorage, 98
MsgListQueryId, 98
MsgListSortType, 99
MsgListStorageRegistry, 99
MsgOpenStoConnection, 100
MsgReleaseQueryIdList, 100
MsgReleaseSortTypeList, 101
MsgReleaseStorageIdList, 101
MsgReleaseStorageRegistry, 102
Msg_Mapi_Storage_Message.h, 103
MsgAddMessage, 105
MsgCountMessage, 105
MsgCountMessageRecursive, 106
MsgCountUnreadMessage, 107
MsgCountUnreadMessageRecursive, 107
MsgDeleteAllMessageInFolder, 108
MsgDeleteMessage, 109
MsgGetMessage, 109

MsgGetMessageCommInfo, 110
MsgGetMessageIdsByStorage, 111
MsgGetMessageParentId, 112
MsgMoveMessageToFolder, 112
MsgMoveMessageToStorage, 113
MsgQueryMessage, 114
MsgQueryMessageRecursive, 115
MsgReleaseMessage, 116
MsgReleaseMessageCommInfo, 116
MsgReleaseMessageIdList, 116
MsgUpdateMessage, 117
Msg_Mapi_Storage_Registry.h, 118
MSG_STORAGE_REGISTRY_S, 118
Msg_Mapi_Types.h, 119
_MSG_CONTACT_TYPE_E, 126
_MSG_ENCODE_TYPE_E, 127
_MSG_ERROR_E, 127
_MSG_FOLDER_TYPE_E, 128
_MSG_MAIN_TYPE_E, 128
_MSG_MEDIA_TYPE_E, 129
_MSG_MESSAGE_PRIORITY_E, 129
_MSG_REQUEST_TYPE_E, 129
_MSG_SESSION_PRIORITY_E, 130
_MSG_SIZE_UNIT_E, 130
_MSG_STATUS_E, 130
_MSG_STORAGE_ID_VALUE_E, 131
bool, 123
false, 123
MSG_ACCOUNT_ID_LIST_S, 124
MSG_ACCOUNT_ID_T, 124
MSG_CONTACT_TYPE_T, 124
MSG_DEFAULT_ACCOUNT_ID, 123
MSG_DEFAULT_FOLDER_ID, 123
MSG_ENCODE_TYPE_T, 124
MSG_ERROR_BEGIN, 123
MSG_ERROR_T, 124
MSG_FOLDER_ID_LIST_S, 124
MSG_FOLDER_ID_T, 125
MSG_FOLDER_TYPE_T, 125
MSG_INVALID_ACCOUNT_ID, 124
MSG_INVALID_FOLDER_ID, 124
MSG_INVALID_MESSAGE_ID, 124
MSG_MAIN_TYPE_T, 125
MSG_MEDIA_TYPE_T, 125
MSG_MESSAGE_ID_LIST_S, 125
MSG_MESSAGE_ID_T, 125
MSG_MESSAGE_PRIORITY_T, 125
MSG_MESSAGE_TYPE_LIST_S, 125
MSG_MESSAGE_TYPE_S, 125
MSG_PLUGIN_ERROR_T, 125
MSG_REQUEST_ID_T, 125
MSG_REQUEST_TYPE_T, 125
MSG_SESSION_PRIORITY_T, 126
MSG_SIZE_UNIT_T, 126
MSG_STATUS_T, 126
MSG_STORAGE_ID_LIST_S, 126
MSG_STORAGE_ID_T, 126
MSG_STRING_S, 126
MSG_SUB_TYPE_T, 126
true, 124
MSG_MEDIA_TYPE_T
Msg_Mapi_Types.h, 125
MSG_MESSAGE_ID_LIST_S
Msg_Mapi_Types.h, 125
MSG_MESSAGE_ID_T
Msg_Mapi_Types.h, 125
MSG_MESSAGE_PRIORITY_T
Msg_Mapi_Types.h, 125
MSG_MESSAGE_REQUEST_S
Msg_Mapi_Message_Request.h, 49
MSG_MESSAGE_S
Msg_Mapi_Message.h, 48
MSG_MESSAGE_STATUS_S
Msg_Mapi_Message_Status.h, 50
MSG_MESSAGE_TYPE_LIST_S
Msg_Mapi_Types.h, 125
MSG_MESSAGE_TYPE_S
Msg_Mapi_Types.h, 125
MSG_PLUGIN_ERROR_T
Msg_Mapi_Types.h, 125
MSG_QUERY_ID_LIST_S
Msg_Mapi_QueryRule.h, 52
MSG_QUERY_ID_T
Msg_Mapi_QueryRule.h, 52
MSG_QUERY_LIST_S
Msg_Mapi_QueryRule.h, 52
MSG_QUERY_S
Msg_Mapi_QueryRule.h, 52
MSG_REQUEST_ID_T
Msg_Mapi_Types.h, 125
MSG_REQUEST_TYPE_T
Msg_Mapi_Types.h, 125
MSG_SESSION_HANDLE_T
Msg_Mapi_Session.h, 55
MSG_SESSION_PRIORITY_T
Msg_Mapi_Types.h, 126
MSG_SIZE_UNIT_T
Msg_Mapi_Types.h, 126
MSG_SORT_RULE_S
Msg_Mapi_QueryRule.h, 52
MSG_SORT_TYPE_LIST_S
Msg_Mapi_QueryRule.h, 52
MSG_SORT_TYPE_T
Msg_Mapi_QueryRule.h, 52
MSG_STATUS_T
Msg_Mapi_Types.h, 126
MSG_STORAGE_ID_LIST_S
Msg_Mapi_Types.h, 126

MSG_STORAGE_ID_T
Msg_Mapi_Types.h, 126
MSG_STORAGE_REGISTRY_S
Msg_Mapi_Storage_Registry.h, 118
MSG_STRING_S
Msg_Mapi_Types.h, 126
MSG_SUB_TYPE_T
Msg_Mapi_Types.h, 126
MsgAddAccount
Msg_Mapi_Storage_Account.h, 67
MsgAddFolder
Msg_Mapi_Storage_Folder.h, 78
MsgAddMessage
Msg_Mapi_Storage_Message.h, 105
MsgCancelReq
Msg_Mapi_Session.h, 56
MsgCloseStoConnection
Msg_Mapi_Storage_Manager.h, 93
MsgConnectSession
Msg_Mapi_Session.h, 57
MsgCountAccount
Msg_Mapi_Storage_Account.h, 68
MsgCountMessage
Msg_Mapi_Storage_Message.h, 105
MsgCountMessageRecursive
Msg_Mapi_Storage_Message.h, 106
MsgCountSubFolder
Msg_Mapi_Storage_Folder.h, 79
MsgCountSubFolderRecursive
Msg_Mapi_Storage_Folder.h, 80
MsgCountUnreadMessage
Msg_Mapi_Storage_Message.h, 107
MsgCountUnreadMessageRecursive
Msg_Mapi_Storage_Message.h, 107
MsgCreateRootFolder
Msg_Mapi_Storage_Folder.h, 80
MsgCreateSession
Msg_Mapi_Session.h, 58
MsgCreateStoConnection
Msg_Mapi_Storage_Manager.h, 94
MsgDeleteAccount
Msg_Mapi_Storage_Account.h, 68
MsgDeleteAllMessageInFolder
Msg_Mapi_Storage_Message.h, 108
MsgDeleteFolder
Msg_Mapi_Storage_Folder.h, 81
MsgDeleteMessage
Msg_Mapi_Storage_Message.h, 109
MsgDestroySession
Msg_Mapi_Session.h, 58
MsgDestroyStoConnection
Msg_Mapi_Storage_Manager.h, 94
MsgDisconnectSession
Msg_Mapi_Session.h, 59
MsgGetAccount
Msg_Mapi_Storage_Account.h, 69
MsgGetAccountIdsByStorage
Msg_Mapi_Storage_Account.h, 70
MsgGetAllRootFolderIds
Msg_Mapi_Storage_Folder.h, 82
MsgGetAllSupportedFilterId
Msg_Mapi_Session.h, 59
MsgGetAllSupportedMsgType
Msg_Mapi_Session.h, 59
MsgGetCommAccount
Msg_Mapi_Storage_Account.h, 71
MsgGetDefaultAccountID
Msg_Mapi_Storage_Account.h, 71
MsgGetFolder
Msg_Mapi_Storage_Folder.h, 83
MsgGetFolderByType
Msg_Mapi_Storage_Folder.h, 84
MsgGetFolderIdsByStorage
Msg_Mapi_Storage_Folder.h, 84
MsgGetFolderParentId
Msg_Mapi_Storage_Folder.h, 85
MsgGetFwStorageStatus
Msg_Mapi_Storage_Manager.h, 95
MsgGetMediaTypeByStorageId
Msg_Mapi_Storage_Manager.h, 96
MsgGetMessage
Msg_Mapi_Storage_Message.h, 109
MsgGetMessageCommInfo
Msg_Mapi_Storage_Message.h, 110
MsgGetMessageIdsByStorage
Msg_Mapi_Storage_Message.h, 111
MsgGetMessageParentId
Msg_Mapi_Storage_Message.h, 112
MsgGetMsgStorageStatus
Msg_Mapi_Storage_Manager.h, 96
MsgGetPredefinedFolders
Msg_Mapi_Storage_Folder.h, 86
MsgGetSessionPluginError
Msg_Mapi_Session.h, 60
MsgGetStoragePluginError
Msg_Mapi_Storage_Manager.h, 97
MsgGetStorageRegistry
Msg_Mapi_Storage_Manager.h, 97
msgId
_MSG_MESSAGE_S, 25
_MSG_MESSAGE_STATUS_S, 29
msgIds
_MSG_MESSAGE_ID_LIST_S, 22
MsgInitStorage
Msg_Mapi_Storage_Manager.h, 98
MsgListQueryId
Msg_Mapi_Storage_Manager.h, 98
MsgListSortType

Msg_Mapi_Storage_Manager.h, 99
MsgListStorageRegistry
Msg_Mapi_Storage_Manager.h, 99
MsgMoveAccountToStorage
Msg_Mapi_Storage_Account.h, 72
MsgMoveFolderToFolder
Msg_Mapi_Storage_Folder.h, 87
MsgMoveFolderToStorage
Msg_Mapi_Storage_Folder.h, 87
MsgMoveMessageToFolder
Msg_Mapi_Storage_Message.h, 112
MsgMoveMessageToStorage
Msg_Mapi_Storage_Message.h, 113
MsgOnMessageIncomingCallback
Msg_Mapi_Session.h, 55
MsgOnStatusChangedCallback
Msg_Mapi_Session.h, 56
MsgOpenStoConnection
Msg_Mapi_Storage_Manager.h, 100
MsgQueryAccount
Msg_Mapi_Storage_Account.h, 73
MsgQueryFolder
Msg_Mapi_Storage_Folder.h, 88
MsgQueryFolderRecursive
Msg_Mapi_Storage_Folder.h, 89
MsgQueryMessage
Msg_Mapi_Storage_Message.h, 114
MsgQueryMessageRecursive
Msg_Mapi_Storage_Message.h, 115
MsgRegMessageCallback
Msg_Mapi_Session.h, 60
MsgRegStatusCallback
Msg_Mapi_Session.h, 61
MsgReleaseAccount
Msg_Mapi_Storage_Account.h, 74
MsgReleaseAccountIdList
Msg_Mapi_Storage_Account.h, 74
MsgReleaseCommAccount
Msg_Mapi_Storage_Account.h, 74
MsgReleaseFilterIdList
Msg_Mapi_Session.h, 61
MsgReleaseFolder
Msg_Mapi_Storage_Folder.h, 90
MsgReleaseFolderIdList
Msg_Mapi_Storage_Folder.h, 90
MsgReleaseMessage
Msg_Mapi_Storage_Message.h, 116
MsgReleaseMessageCommInfo
Msg_Mapi_Storage_Message.h, 116
MsgReleaseMessageIdList
Msg_Mapi_Storage_Message.h, 116
MsgReleaseMsgTypeList
Msg_Mapi_Session.h, 62
MsgReleaseQueryIdList
Msg_Mapi_Storage_Manager.h, 100
MsgReleaseSortTypeList
Msg_Mapi_Storage_Manager.h, 101
MsgReleaseStorageIdList
Msg_Mapi_Storage_Manager.h, 101
MsgReleaseStorageRegistry
Msg_Mapi_Storage_Manager.h, 102
MsgSetDefaultAccountID
Msg_Mapi_Storage_Account.h, 74
MsgSetFilterList
Msg_Mapi_Session.h, 62
MsgSetPriority
Msg_Mapi_Session.h, 63
MsgSubmitReq
Msg_Mapi_Session.h, 63
msgType
_MSG_ACCOUNT_S, 10
_MSG_MESSAGE_REQUEST_S, 23
_MSG_MESSAGE_S, 25
MsgUpdateAccount
Msg_Mapi_Storage_Account.h, 75
MsgUpdateFolder
Msg_Mapi_Storage_Folder.h, 90
MsgUpdateMessage
Msg_Mapi_Storage_Message.h, 117

name
_MSG_STORAGE_REGISTRY_S, 39

parentId
_MSG_FOLDER_S, 20
_MSG_MESSAGE_S, 25
path
_MSG_STORAGE_REGISTRY_S, 39
pCommExtData
_MSG_FOLDER_S, 21
_MSG_MESSAGE_S, 27
pContacts
_MSG_CONTACT_LIST_S, 12
pExtData
_MSG_ACCOUNT_S, 11
_MSG_CONTACT_S, 13
_MSG_MESSAGE_REQUEST_S, 24
_MSG_MESSAGE_S, 27
_MSG_MESSAGE_STATUS_S, 30
pMsg
_MSG_MESSAGE_REQUEST_S, 23
pMsgTypes
_MSG_MESSAGE_TYPE_LIST_S, 31
priority
_MSG_MESSAGE_S, 27
pStorageIds
_MSG_STORAGE_ID_LIST_S, 38
pString

_MSG_STRING_S, 40

queries
_MSG_QUERY_LIST_S, 34
queryId
_MSG_QUERY_S, 35
queryIds
_MSG_QUERY_ID_LIST_S, 33
queryValue
_MSG_QUERY_S, 35

reqId
_MSG_MESSAGE_REQUEST_S, 23
_MSG_MESSAGE_STATUS_S, 29
reqType
_MSG_MESSAGE_REQUEST_S, 23
rootFolderId
_MSG_ACCOUNT_S, 10

size
_MSG_MESSAGE_S, 27
sortType
_MSG_FOLDER_S, 20
_MSG_SORT_RULE_S, 36
sortTypes
_MSG_SORT_TYPE_LIST_S, 37
status
_MSG_MESSAGE_S, 27
_MSG_MESSAGE_STATUS_S, 29
storageId
_MSG_ACCOUNT_S, 10
_MSG_FOLDER_S, 20
_MSG_MESSAGE_S, 26
storeId
_MSG_STORAGE_REGISTRY_S, 39
subject
_MSG_MESSAGE_S, 26
subType
_MSG_MESSAGE_TYPE_S, 32

to
_MSG_MESSAGE_S, 26
true
Msg_Mapi_Types.h, 124

valueLength
_MSG_FILTER_S, 18
_MSG_QUERY_S, 35